

Institut für Parallele und Verteilte Systeme

Universität Stuttgart  
Universitätsstraße 38  
D-70569 Stuttgart

Bachelorarbeit

# **Automatisierte Integration von IoT-Geräten in IoT-Umgebungen**

Maximilian Reichel

**Studiengang:** Informatik

**Prüfer/in:** Prof. Dr.-Ing. habil. Bernhard Mitschang

**Betreuer/in:** Daniel Del Gaudio, M.Sc.

**Beginn am:** 11. April 2019

**Beendet am:** 5. November 2019



## Kurzfassung

Das Internet of Things (dt. Internet der Dinge, kurz IoT) gewinnt seit einiger Zeit rasch an Bedeutung. Immer mehr Objekte der realen Welt sind in der Lage über Standard-Internetprotokolle miteinander zu kommunizieren. Da diese IoT-Geräte typischerweise Sensoren und Aktoren besitzen, können sie Informationen über ihre Umgebung erfassen, Situationen erkennen und die reale Welt direkt beeinflussen. In IoT-Umgebungen wie Smart Homes können sie dadurch das alltägliche Leben der Menschen erleichtern. Auch in Smart Citys und der Industrie gibt es vielseitige Anwendungsmöglichkeiten. Damit diese Geräte Informationen untereinander austauschen können, um gemeinsame Ziele zu erreichen, muss auf den Geräten die korrekte Software installiert sein und sie müssen füreinander eingerichtet werden. Aufgrund der steigenden Anzahl von IoT-Geräten stellt es eine Herausforderung dar, all diese Geräte zu verwalten und einzurichten. Eine manuelle Einrichtung ist sehr zeitaufwendig und fehleranfällig. Zudem bestehen IoT-Umgebungen meist aus heterogenen Geräten mit unterschiedlichen Fähigkeiten, sodass je nach Gerät andere Software auf den Geräten benötigt wird. Deshalb werden Lösungen benötigt, die eine große Anzahl von heterogenen IoT-Geräten automatisiert verwalten und abhängig von den Fähigkeiten korrekt einrichten können. In dieser Bachelorarbeit wird ein Konzept vorgestellt, um neue Geräte automatisiert in eine IoT-Umgebung zu integrieren und bestehende Geräte zu verwalten und zu überwachen. Eine zentrale Serverkomponente dient als Registrierungsserver für neue Geräte. Anhand deren Eigenschaften wird die passende Software ermittelt und installiert. Zudem überwacht die zentrale Serverkomponente die Geräte und passt deren Konfiguration kontinuierlich auf die sich ändernden Gegebenheiten der Umgebung an.



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>13</b>
1.1	Motivation . . . . .	14
<b>2</b>	<b>Grundlagen</b>	<b>17</b>
2.1	Netzwerkkommunikation . . . . .	17
2.2	Messaging Engine . . . . .	18
2.3	Constrained Application Protocol . . . . .	19
2.4	Multi-purpose Binding and Provisioning Platform . . . . .	21
<b>3</b>	<b>Verwandte Arbeiten</b>	<b>23</b>
<b>4</b>	<b>Konzept</b>	<b>27</b>
4.1	Überblick . . . . .	27
4.2	Schritt 1: Server-Discovery . . . . .	29
4.3	Schritt 2: Registrierung . . . . .	32
4.4	Schritt 3: Softwareauswahl . . . . .	33
4.5	Schritt 4: Software-Deployment . . . . .	36
4.6	Schritt 5: Konfiguration aller Geräte . . . . .	37
4.7	Schritt 6: Datenverarbeitung und Monitoring . . . . .	40
4.8	Schritt 7: Geräte entfernen . . . . .	42
4.9	Mögliche Optimierungen . . . . .	43
<b>5</b>	<b>Implementierung</b>	<b>47</b>
5.1	CDM-Agent . . . . .	48
5.2	CDM-Server . . . . .	51
<b>6</b>	<b>Zusammenfassung und Ausblick</b>	<b>61</b>
	<b>Literaturverzeichnis</b>	<b>63</b>



# Abbildungsverzeichnis

1.1	Symbolische Abbildung eines Smart Homes . . . . .	14
2.1	Veranschaulichung verschiedener Kommunikationsformen . . . . .	17
2.2	Beispiel für ein als Graph dargestelltes Datenflussmodell einer Raumtemperaturregelung . . . . .	19
4.1	Konzeptübersicht . . . . .	28
4.2	Konzeptionelle Darstellung der Registrierung eines neuen Geräts . . . . .	29
4.3	Grafische Darstellung der Multicast-Discovery . . . . .	30
4.4	Graph des Datenflussmodells für das verwendete Beispiel . . . . .	38
4.5	Die Teilgraphen $DF_d$ des Datenflussmodells, die die Geräte aus dem Beispiel jeweils erhalten . . . . .	39
4.6	Veranschaulichung der Datenverarbeitung mithilfe der Messaging Engine . . . . .	40
4.7	Grafische Darstellung des Ablaufs: Der CDM-Server erhält eine Heartbeat-Nachricht von einem unbekanntem Gerät . . . . .	41
5.1	Übersicht der Architektur des CDM-Servers . . . . .	51
5.2	UML-Diagramm von einem Geräte-Event . . . . .	52
5.3	Grafische Darstellung des Event-Austauschs zwischen den Komponenten des CDM-Servers . . . . .	52
5.4	Datenflussmodell aus Listing 5.2 als Graph dargestellt . . . . .	54





## Tabellenverzeichnis

- 4.1 Tabellarische Darstellung der Informationen, die die Geräte über andere Geräte bzw. deren Operationen unter Anwendungen der beschriebenen Optimierungen besitzen 38

## Verzeichnis der Listings

- 5.1 Beispielhafte Registrierungsanfrage in JSON-Repräsentation . . . . . 50
- 5.2 Beispielhaftes Datenflussmodell in JSON-Repräsentation . . . . . 54

## Verzeichnis der Algorithmen

- 4.1 Greedy-Algorithmus zur Bestimmung der Operationen für ein Gerät . . . . . 36



# Abkürzungsverzeichnis

- CBOR** Concise Binary Object Representation. 47
- CDM** Constraint Device Management. 27
- CoAP** Constrained Application Protocol. 19
- CoAP RD** CoAP Resource Directory. 23
- DHCP** Dynamic Host Configuration Protocol. 31
- DNS** Domain Name System. 32
- FIFO** First-In-First-Out. 56
- HTTP** Hypertext Transfer Protocol. 20
- IGMP** Internet Group Management Protocol. 18
- IoT** Internet of Things. 13
- JSON** JavaScript Object Notation. 47
- MBP** Multi-purpose Binding and Provisioning Platform. 21
- ME** Messaging Engine. 18
- MQTT** Message Queuing Telemetry Transport. 21
- SNMP** Simple Network Management Protocol. 41
- SSDP** Simple Service Discovery Protocol. 24
- SSH** Secure Shell. 21
- TCP** Transmission Control Protocol. 20
- UDP** User Datagram Protocol. 20
- UPnP** Universal Plug and Play. 24
- URL** Uniform Resource Locator. 20
- VM** virtuelle Maschine. 47



# 1 Einleitung

Das Internet of Things (dt. Internet der Dinge, kurz IoT) [VF13] gewinnt seit einiger Zeit rasch an Bedeutung. Heterogene Geräte tauschen Daten über Standard-Internetprotokolle aus, um gemeinsame Ziele zu erreichen. Dabei wird für die Übertragung der Daten keine menschliche Interaktion benötigt. Angefangen bei der intelligenten Glühbirne, die sich mithilfe des Smartphones steuern lässt, bis hin zu Sensoren und Aktoren in komplexen Industrieanlagen, sind die Anwendungsmöglichkeiten vielseitig. Durch Sensoren kann die IoT-Umgebung selbstständig Informationen über das Umfeld sammeln und bei Bedarf entsprechende Aktionen ausführen um, zum Beispiel, auf neue Situationen zu reagieren. Bei steigender Anzahl an Geräten in IoT-Umgebungen, wird es immer aufwendiger neue Geräte zu integrieren. Unter Umständen muss die Konfiguration nicht nur an dem zu integrierenden Gerät vorgenommen werden, sondern auch an anderen Komponenten der Umgebung. Dadurch kann eine manuelle Konfiguration der Geräte in komplexen, unübersichtlichen Systemen fehleranfällig und arbeitsintensiv sein, was zu einer eingeschränkten Skalierbarkeit führt [GRM17].

Da der Begriff Internet of Things noch relativ jung ist, gibt es noch keine eindeutige Definition dafür. Atzori et al. beschreiben die Idee vom Internet of Things mit der zunehmenden Präsenz einer Vielzahl von Dingen wie Mobiltelefonen, Sensoren und Aktoren, die eindeutig adressierbar sind, miteinander interagieren können und so durch Zusammenarbeit gemeinsame Ziele erreichen [AIM10]. Bekannte IoT-Anwendungsgebiete sind etwa Smart Home, Smart City und die Industrie. In Smart Homes können IoT-Geräte die Bewohner im Alltag unterstützen. Beispielweise kann die Heizung entsprechend den Gewohnheiten der Bewohner gesteuert werden. Auch im Bereich des Gesundheitswesens ermöglichen mit Sensoren ausgestattete, internetfähige Geräte einige Anwendungsszenarien. So haben Ishii et al. ein Konzept zur Früherkennung von Demenz mithilfe von IoT entwickelt [IKA+16]. Außerdem stellen Greene et al. ein IoT-basiertes System zur Sturzerkennung vor, das Hilfe verständigt und mit Betroffenen kommunizieren kann [GTC16]. In Smart Citys kann die Sicherheit auf den Straßen mittels intelligenter Straßenbeleuchtung verbessert werden. Knobloch und Braunschweig [KB17] haben ein Konzept vorgestellt, das dafür sorgt, dass die Straßenbeleuchtung nur nach Bedarf eingeschaltet wird, abhängig von Ort und Bewegung einer Person oder eines Fahrzeugs. Durch diese Art der Beleuchtung kann der Energieverbrauch im Vergleich zur klassischen Straßenbeleuchtung noch weiter reduziert werden. Ebenso können intelligente IoT-Lösungen in der Industrie eingesetzt werden, wie beispielsweise bei Warenauslieferung, Logistik und Produktion. In der Warenlogistik können Informationen über die Position von Waren kontaktlos und in Echtzeit erfasst werden, wodurch große Warenmengen einfach koordiniert werden können [YJX10]. IoT kann ebenso in der Wartung von Maschinen eingesetzt werden. So hat Selcuk [Sel17] Ansätze vorgestellt, um Messdaten von Sensoren an Maschinen so auszuwerten, dass diese auf die Notwendigkeit einer Wartung hinweisen.

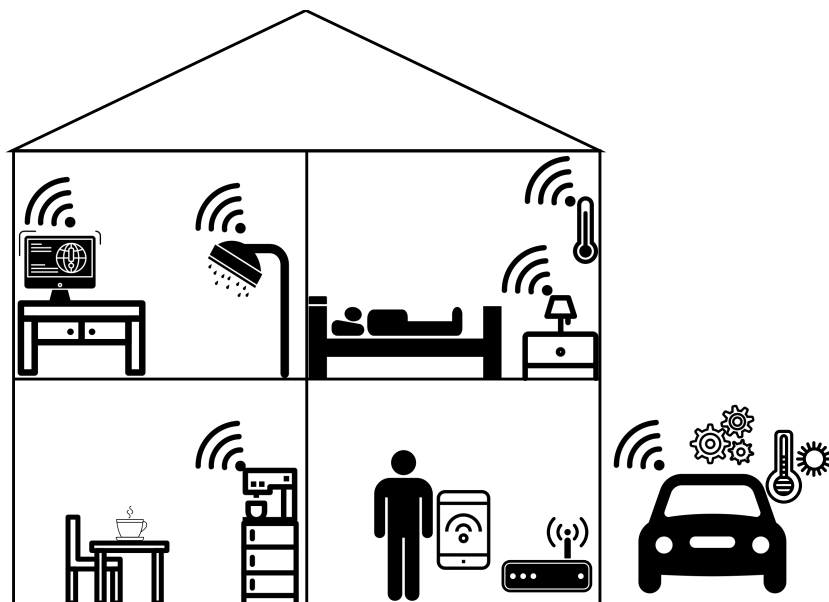
Doch die vielen Möglichkeiten der IoT bergen auch Risiken in Bezug auf die Privatsphäre der Menschen. So ist es nicht immer offensichtlich, welche Geräte welche Daten erheben und wohin diese Daten übertragen werden. Besonders kritisch sind dabei Aufnahmen von Sprachassistenten

oder Überwachungskameras, da diese persönliche Details enthalten können [BJD16]. Kameraaufzeichnungen können mit Techniken wie Gesichts- oder Objekterkennung automatisch analysiert werden, was allerdings sehr rechenintensiv ist. Edge Computing [SCZ+16] bezeichnet ein Paradigma, das Ressourcen von Rechnern am Rand des Netzwerks zur Datenverarbeitung verwendet. Die Auslagerung von Berechnungen zur Videoanalyse auf Edge-Knoten stellt ein typisches Anwendungsszenario für Edge Computing dar.

### 1.1 Motivation

Smart Homes [Har03] sollen die Bewohner im Alltag unterstützen und den Komfort erhöhen. Gleichzeitig können Kosten durch optimierte Energienutzung eingespart werden. Um die Bewohner im Alltag unterstützen zu können, muss das System die Gewohnheiten der Bewohner kennen und deren Zusammenhänge feststellen. Cook et al. beschreiben ein mögliches Smart-Home-Szenario in „MavHome: an agent-based smart home“ [CYH+03] folgendermaßen:

Ein Smart Home ist ein intelligentes System, das von den Bewohnern des Hauses lernt. Bob wohnt in einem Smart Home (siehe Abbildung 1.1). Zum Beispiel kennt das System die morgendliche Routine des Bewohners Bob und schaltet um 6:45 Uhr die Heizung an, damit die Wohnung die perfekte Temperatur hat, wenn das System um 7 Uhr den Wecker klingeln lässt und das Licht im Schlafzimmer einschaltet. Bob kann jetzt in ein angenehm temperiertes Bad gehen. Wenn er dort das Licht einschaltet, erscheinen auf einem Bildschirm die neuesten Nachrichten und die Dusche geht an. Wenn Bob im Bad fertig ist, geht das Licht aus, der Bildschirm wird dunkel und alles verlagert sich in die Küche. Dort geht das Licht an, der erste Kaffee ist schon fertig und die Nachrichten sind nun hier auf einem Bildschirm zu sehen. Dieses Szenario ließe sich wie folgt



**Abbildung 1.1:** Symbolische Abbildung eines Smart Homes

erweitern: Während Bob frühstückt schaltet das Smart Home noch rechtzeitig die Standheizung in Bobs selbstfahrendem Auto an. Um diese Zusammenhänge zu ermitteln wird viel Rechenleistung benötigt. Bobs selbstfahrendes Auto besitzt große Mengen an Rechenleistung, die während der Fahrt benötigt werden. Wenn das Fahrzeug nachts in Bobs Einfahrt steht und zum Aufladen an das Stromnetz angeschlossen ist, benötigt es diese Ressourcen nicht selbst. Die Daten, die das Smart Home über den Tag von leistungsschwachen Geräten wie Lichtschaltern, Temperatursensoren, etc. über Bob und sein Umfeld erhebt, können nachts mithilfe der Ressourcen seines Autos ausgewertet werden. Dazu installiert das Smart Home in dieser Zeit eigene Applikationen auf dem Auto, um maschinelles Lernen auf die Daten anzuwenden und Zusammenhänge zu finden.

Damit diese Geräte zusammenarbeiten können, müssen sie füreinander konfiguriert werden und auf jedem Gerät muss die entsprechende Software installiert sein. Die Geräte sollen aber das Leben der Menschen erleichtern und nicht weiter verkomplizieren, weshalb eine manuelle Einrichtung der Geräte nicht zumutbar ist. Ortsveränderliche Geräte wie Bobs Auto oder sein Smartphone sind immer nur zeitweise Teil der Umgebung, wodurch die anderen Geräte dynamisch ständig neu konfiguriert werden müssen. Händisch wäre diese oft wiederholte Konfiguration nicht realisierbar, zumal viele Nutzer damit überfordert wären.

Ziel dieser Arbeit ist es daher, ein Konzept zu entwickeln, das die automatisierte Integration solcher Geräte in einer IoT-Umgebung ermöglicht, sodass kein manueller Nutzereingriff erforderlich ist. Dazu soll in der Umgebung ein Server zur Verwaltung und Integration der Geräte verwendet werden. Die Geräte sollen diesen Server selbstständig finden und sich daran anmelden. Der Server soll dann dafür sorgen, dass die richtige Software auf den Geräten installiert wird und die Geräte eingerichtet werden. Außerdem soll der Server die Anwesenheit der Geräte überwachen, sodass er erkennt, wenn Geräte die Umgebung verlassen haben, um dann die anderen Geräte entsprechend zu konfigurieren.

Die weitere Arbeit ist wie folgt gegliedert:

In Kapitel 2 werden die notwendigen Grundlagen erläutert. Kapitel 3 gibt einen Einblick in andere Publikationen, die in Zusammenhang mit dieser Bachelorarbeit stehen. In Kapitel 4 wird das erarbeitete Konzept dieser Bachelorarbeit beschrieben und bildet somit den Kern dieser Arbeit. In Kapitel 5 wird auf die prototypische Implementierung des Konzepts eingegangen. Zuletzt fasst Kapitel 6 die Ergebnisse dieser Arbeit zusammen und gibt einen kurzen Ausblick.



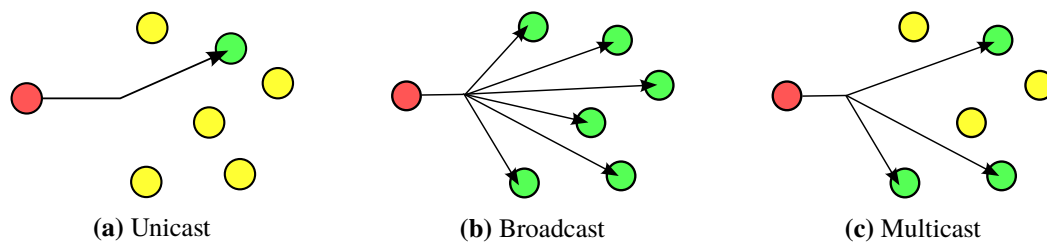


## 2 Grundlagen

In diesem Kapitel werden die Grundlagen erläutert, die zum Verständnis der Arbeit benötigt werden. Zu Beginn werden drei in IP-Netzwerken gängige Kommunikationsformen vorgestellt. Anschließend werden zwei Programme beschrieben, die Multi-purpose Binding and Provisioning Platform und die Messaging Engine, die für das Konzept dieser Arbeit verwendet werden. Die ME spielt für die Arbeit eine entscheidende Rolle. Schließlich wird das für das Konzept dieser Arbeit verwendete Kommunikationsprotokoll CoAP vorgestellt.

### 2.1 Netzwerkkommunikation

Teilnehmer eines Netzwerks können verschiedene Kommunikationsformen verwenden, um miteinander zu kommunizieren. In diesem Abschnitt werden drei typische Kommunikationsformen vorgestellt. Diese sind in Abbildung 2.1 grafisch dargestellt. Der rote Kreis stellt jeweils den Sender dar, die grünen Kreise die Empfänger und die Pfeile die Nachrichten. Überwiegend werden Pakete in einem IP-Netzwerk von einem Gerät an einen Empfänger adressiert. Dieses Paket wird dann genau zu diesem einen Empfänger geleitet (siehe Abbildung 2.1a). Die Adressierung einer Nachricht an einen einzigen Empfänger wird als Unicast-Kommunikation bezeichnet. Dabei ist es notwendig, dass dem Absender die IP-Adresse des Empfängers bekannt ist. Im Vergleich dazu gibt es auch Multicast- und Broadcast-Kommunikation. Hierbei wird eine Nachricht nicht nur an einen einzigen Empfänger, sondern an viele Empfänger zugestellt. Bei Broadcast-Kommunikation wird die Nachricht an ein Netzwerk adressiert, sodass alle Teilnehmer des Netzwerks diese erhalten (siehe Abbildung 2.1b). Dadurch können Netzwerkteilnehmer miteinander kommunizieren, ohne dabei explizit die IP-Adressen der Kommunikationspartner kennen zu müssen. Bei der Broadcast-Kommunikation erhalten allerdings auch diejenigen Netzwerkteilnehmer die Nachrichten, die an dieser Kommunikation nicht interessiert sind, wodurch sie überflüssig belastet werden. Möchten nur bestimmte Teilnehmer untereinander auf diese Weise kommunizieren, kann dazu Multicast-Kommunikation verwendet werden. Dabei tritt jeder Teilnehmer einer oder mehreren Multicast-Gruppen bei, wobei jede Gruppe eine Multicast-Adresse besitzt. Eine Nachricht, die an die Adresse einer solchen



**Abbildung 2.1:** Veranschaulichung verschiedener Kommunikationsformen  
(Quelle: <https://commons.wikimedia.org/>)

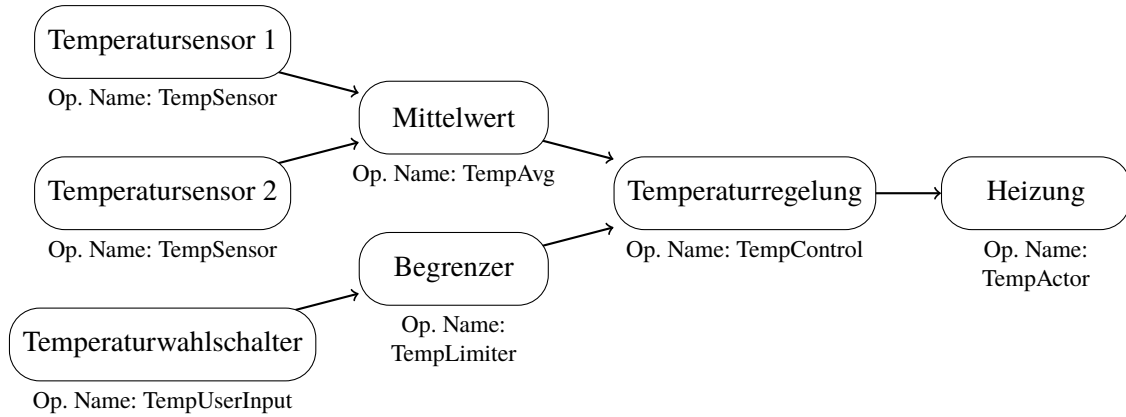
Multicast-Gruppe gesendet wird, erreicht alle Teilnehmer der Gruppe (siehe Abbildung 2.1c). In IP-Netzwerken kann zum Beispiel das Internet Group Management Protocol (IGMP) [CDF+02] dazu verwendet werden, damit das Netzwerk Nachrichten an eine Multicast-Gruppe tatsächlich nur an die Teilnehmer überträgt, die sich für die Nachrichten an die jeweilige Gruppe interessieren. Dazu publizieren die Teilnehmer im Netzwerk, an welchen Multicast-Gruppen sie interessiert sind. Router und Switches, die das IGMP unterstützen, leiten Nachrichten an eine Multicast-Gruppe nur an die Netzwerksegmente weiter, in denen sich Teilnehmer befinden, die an Nachrichten für diese Gruppe interessiert sind.

## 2.2 Messaging Engine

Del Gaudio und Hirmer [DH19] haben ein Konzept entwickelt, um Sensordaten von IoT-Geräten in einer IoT-Umgebung dezentral und möglichst auf den Geräten zu verarbeiten, ohne dass eine zentrale Instanz benötigt wird. In ihrem Konzept wird eine sogenannte Messaging Engine (ME) vorgestellt, die auf den IoT-Geräten installiert ist und für die Verarbeitung der Daten sorgt. Um den Ablauf der Datenverarbeitung zu beschreiben, werden sogenannte Datenflussmodelle eingesetzt, die eine Form des Pipes und Filter-Architekturmusters darstellen [BL98]. Das Pipes und Filter-Architekturmuster kennt zwei Entitäten: Pipes (dt. Rohre) und Filter. Filter wenden bestimmte Operationen auf Daten an und können Ein- und Ausgänge haben. Durch die Pipes können die Ein- und Ausgänge verschiedener Filter miteinander verbunden werden. Ein Filter verarbeitet die eingehenden Daten und leitet die Produkte davon entlang der ausgehenden Pipes an andere Filter weiter. Datenflussmodelle legen fest, welche dieser Filteroperationen mit welchen anderen verbunden sind und beschreiben damit in welcher Reihenfolge die Operation auf die zu verarbeitenden Daten angewendet werden sollen. Dabei ist es auch möglich, dass sich ein Datenfluss in parallele Flüsse aufteilt und später wieder zusammenfließt. Dadurch können auch komplexere Szenarien modelliert werden.

Del Gaudio und Hirmer verwenden gerichtete Graphen, um die Datenflussmodelle zu beschreiben. Ein Graph ist eine abstrakte Struktur, um die Verbindungen zwischen Objekten darzustellen. Dazu besitzt ein Graph zwei Mengen: Eine Menge mit sogenannten Knoten, die eine mathematische Abstraktion der Objekte darstellen und eine Menge mit sogenannten Kanten, die die Verbindungen zwischen den Knoten angeben und so die Verbindungen zwischen den Objekten repräsentieren. In dem Graphen eines Datenflussmodells stellen die Knoten des Graphen die Filter des Pipes und Filter-Architekturmusters dar und die Pipes sind die Kanten des Graphen. In Abbildung 2.2 ist der Graph eines Datenflussmodells für eine Raumtemperaturreglung dargestellt. Die Kreise sind dabei die Knoten des Graphen und die Pfeile zwischen den Knoten stellen die Kanten dar. Der Pfeil zeigt dabei in die Richtung, in die die Daten übertragen werden. Dieses Datenflussmodell beschreibt eine Raumtemperaturreglung mit zwei Temperatursensoren und einem Temperaturwahlschalter. Von den Werten der beiden Sensoren wird der arithmetische Mittelwert zur Regelung verwendet. Die Werte des Temperaturwahlschalters werden auf ein festgelegtes Maximum bzw. Minimum begrenzt. Anschließend wird anhand dieser Werte entschieden, ob es nötig ist zu heizen.

Die Filteroperationen werden im Folgenden als Operationen bezeichnet und ein Gerät auf dem die ME ausgeführt wird, wird im Folgenden als Knoten bezeichnet. Jeder Knoten hat die Fähigkeiten bestimmte Operationen auszuführen. Die ME auf jedem Knoten hat Informationen über die Operationen, die der Knoten selbst ausführen kann, und hat diese Informationen über andere Knoten der Umgebung. Ein Knoten verarbeitet eingehende Daten nach den Definitionen des



**Abbildung 2.2:** Beispiel für ein als Graph dargestelltes Datenflussmodell einer Raumtemperaturregelung

Datenflussmodells und seinen Fähigkeiten. Hat ein Knoten Daten produziert, die er selbst nicht weiterverarbeiten kann, so leitet die ME die Daten an einen anderen Knoten weiter, der dazu in der Lage ist, die nächste Operation auszuführen.

Del Gaudio und Hirmer haben Datenflussmodelle in [DH19] als gerichteten Graph wie folgt formalisiert:

$$DF = (O, E) \quad (2.1)$$

mit der Menge der Operationen

$$O = \{o_0, \dots, o_n\} \quad (2.2)$$

und der Menge der Kanten

$$E \subseteq O \times O \quad (2.3)$$

wobei jeder Knoten eine Operation darstellt und jede Kante  $(o_i, o_j)$  bedeutet, dass die von der Ausführung der Operation  $o_i$  resultierenden Daten als Eingaben von Operation  $o_j$  verwendet werden. Ebenso definieren Del Gaudio und Hirmer eine Funktion, mit der für eine Operation alle nachfolgenden Operationen für diese im Datenflussmodell bestimmt werden können:

$$\begin{aligned} suc: O &\rightarrow \mathcal{P}(O) \\ a &\mapsto \{b \in O \mid (a, b) \in E\} \end{aligned} \quad (2.4)$$

Dabei bezeichnet  $\mathcal{P}(O)$  die Potenzmenge von  $O$ . Jede Operation besitzt einen Namen, der zu der jeweiligen Definition der Operation gehört. So kann ein Datenflussmodell mehrere Operationen enthalten, die denselben Namen haben.

## 2.3 Constrained Application Protocol

Das Constrained Application Protocol (CoAP) ist ein leichtgewichtiges Web-Kommunikationsprotokoll, welches für die Kommunikation zwischen ressourcenbeschränkten Geräten über Netzwerke mit beschränkter Leistung oder Zuverlässigkeit konzipiert wurde. Ein Beispiel dafür sind

solche Geräte mit 8-Bit-Mikrocontroller, welche in der Regel nur über einen geringen Programm- und Arbeitsspeicher verfügen [BCS12]. In vielen Web-Anwendungen wird das Hypertext Transfer Protocol (HTTP) zum Austausch von Informationen zwischen Endpunkten eingesetzt. Ein Endpunkt kann dabei entweder die Server- oder Client-Rolle einnehmen. Der Endpunkt, der Ressourcen oder Dienste für andere anbietet, wird Server genannt. Der Endpunkt, der diese Ressourcen oder Dienste verwendet, wird Client genannt. Da sich CoAP ebenso wie HTTP an das REST-Paradigma [FT00] hält, besitzen die Protokolle einige Gemeinsamkeiten: Beide Protokolle verwenden Uniform Resource Locators (URLs) [BFM05] zur Adressierung von Ressourcen, beide Protokolle benutzen die Anfragemethoden GET, POST, PUT und DELETE, um mit den Ressourcen zu arbeiten und beide Protokolle sind zustandslos. HTTP wird allerdings schon seit über zwei Jahrzehnten ständig weiterentwickelt, wodurch der Umfang des Protokolls inzwischen sehr groß ist. Dies führt dazu, dass die Implementierung von HTTP sehr aufwendig ist und auf ressourcenbeschränkten Geräten verhältnismäßig viel Programmspeicher belegt. Im Gegensatz dazu ist CoAP für die Implementierung auf solchen Geräten optimiert. Eine Nachricht besteht aus dem sogenannten Header und den Nutzdaten. Der Header enthält Metainformationen, wie die Art der Nachricht, die Sender-Adresse, Empfänger-Adresse und weitere Optionen. Um die Größe des Headers zu reduzieren und damit die Menge der Daten zu minimieren, die zusätzlich zu den Nutzdaten über das Netzwerk übertragen werden müssen, verwendet CoAP, im Gegensatz zu HTTP, komprimierte Header.

Zum Übertragen von Nachrichten verwendet HTTP das Transmission Control Protocol (TCP), ein verbindungsorientiertes Protokoll. Das bedeutet, TCP stellt zum Übertragen der Daten eine Verbindung zwischen Sender und Empfänger her, wobei TCP die Integrität der Daten sicherstellt, die über diese Verbindung übertragen werden. Zum Aufbauen der Verbindung vor bzw. zum Abbauen der Verbindung nach der Datenübertragung müssen zusätzliche Daten zwischen Sender und Empfänger ausgetauscht werden. Wird TCP verwendet, um nur geringe Datenmengen zu übertragen, was typisch in der IoT ist, kann die für Verbindungsauf- und abbau zusätzlich übertragene Datenmenge ein Vielfaches höher sein, als die tatsächlich über die Verbindung übertragene Datenmenge. Im Vergleich dazu verwendet CoAP das User Datagram Protocol (UDP), ein verbindungsloses Übertragungsprotokoll. UDP baut keine Verbindung für die Übertragung der Daten auf, wodurch die bei TCP beschriebenen Zusatzdaten vermieden werden. Zusätzlich wird die Zeit des Verbindungsaufbaus eingespart, daher werden die ersten Daten bereits nach kürzerer Zeit übertragen. Dagegen besitzt UDP im Vergleich zu TCP keine Möglichkeit, den Verlust von Daten im Netzwerk festzustellen, weshalb CoAP eigene Mechanismen definiert, um die Übertragung einer Nachricht sicherzustellen.

Im Gegensatz zu HTTP bietet CoAP auch die Möglichkeit der Multicast-Kommunikation. Dabei können einzelne Nachrichten gleichzeitig an mehrere Empfänger adressiert werden. Dies wäre bei Verwendung von TCP nicht möglich.

Da CoAP auch für den automatisierten Informationsaustausch zwischen Endpunkten entwickelt wurde, bietet CoAP Mechanismen, die es jedem Endpunkt ermöglicht, andere Endpunkte in einem Netzwerk zu finden und Informationen über die von diesen angebotenen Diensten bzw. Ressourcen zu erhalten. Die Eigenschaften dieser Ressourcen werden einheitlich mithilfe des CoRE Link Format beschrieben. Das CoRE Link Format [RFC6690] beschreibt Ressourcen und deren Eigenschaften und Beziehungen zu anderen Ressourcen. Dabei werden den Ressourcen Attribute zugeordnet, über die sich Semantik und Verhalten der Ressource eindeutig feststellen lassen. Besitzt beispielsweise eine Ressource das Typ-Attribut *oic.r.temperature*, so handelt es sich dabei um eine Schnittstelle, an der der Wert eines Temperatursensors abgefragt werden kann.

Außerdem besitzt CoAP auch die Möglichkeit, Ressourcen zu beobachten und so über Änderungen

sofort informiert zu werden. Wenn ein Client ständig über den aktuellen Zustand einer Ressource informiert sein möchte, kann er dies dem Server mitteilen. Der Server benachrichtigt dann den Client, wenn sich der Zustand geändert hat. Ein typischer Anwendungsfall wäre das Auslesen des Werts eines Temperatursensors. Durch diese Möglichkeit zur Beobachtung von Ressourcen muss der Client die Ressource nicht zyklisch abfragen, um Änderungen festzustellen. Durch die vielen Gemeinsamkeiten von CoAP und HTTP ist es außerdem möglich, CoAP-Anfragen auf das HTTP abzubilden, was eine einfache Integration von CoAP-Applikationen in bestehende Webapplikationen ermöglicht.

## 2.4 Multi-purpose Binding and Provisioning Platform

Die Multi-purpose Binding and Provisioning Platform (MBP) [FHPM18] [HBS+16] ist eine Plattform, um IoT-Geräte zu überwachen und zu integrieren, damit die Sensoren und Aktoren der Geräte genutzt und mit Anwendungen verknüpft werden können. Die MBP zeichnet sich durch eine grafische Benutzeroberfläche aus, über die die IoT-Geräte und die benötigte Software verwaltet werden. Diese Softwareartefakte sind in Form von sogenannten Adaptern in der MBP hinterlegt. Diese Adapter werden von der MBP auf den Geräten installiert und ermöglichen es der MBP Aktoren zu steuern und Sensordaten der Geräte zu nutzen bzw. für andere Applikationen, zum Beispiel über das Message Queuing Telemetry Transport (MQTT) Protokoll [BG14] und HTTP, bereitzustellen. MQTT ist ein Protokoll, das mithilfe eines MQTT-Brokers den Nachrichtenaustausch zwischen Maschinen ermöglicht. Dabei ist jede Nachricht einem sogenannten Topic (dt. Thema) zugeordnet. Am MQTT-Broker abonnieren die Clients die Topics, für die sie Nachrichten erhalten wollen. Eine Nachricht wird an den MQTT-Broker gesendet, welcher diese dann an die Clients übermittelt, die das zugehörige Topic abonniert haben. Das Installieren der Software auf den Geräten erfolgt via dem Secure Shell (SSH)-Protokoll [LY06] über Skripte, die in den Adaptern hinterlegt sein müssen. Per SSH kann eine sichere Verbindung über ein Netzwerk zu einem entfernten Computer aufgebaut werden. Über diese Verbindung können dann unter anderem Befehle auf der Kommandozeile des entfernten Computers ausgeführt oder Dateien übertragen werden. Zum Aufbau der Verbindung werden entsprechende Zugangsdaten benötigt. Jeder Adapter benötigt beispielsweise ein "install.sh"-Skript für die Installation der Software und ein "start.sh"-Skript, um die Software anschließend zu starten. Bei der Registrierung eines Geräts ist es daher notwendig, entsprechende Anmeldeinformationen für den Zugriff auf das Gerät zu hinterlegen. Des Weiteren wird dem Gerät ein eindeutiger Name zugewiesen und weitere Informationen, wie zum Beispiel die Netzwerkadresse, hinterlegt.



## 3 Verwandte Arbeiten

CoAP Resource Directory (CoAP RD) ist eine Erweiterung für CoAP, die sich derzeit noch in der Entwicklung befindet [SKB+19]. An CoAP RDs können CoAP-Endpunkte die von ihnen angebotenen Ressourcen registrieren und dort die registrierten Ressourcen der anderen Geräte durchsuchen. Sind für Ressourcen die im CoRE Link Format festgelegten Attribute wie Typ der Ressource oder Schnittstellenbeschreibung angegeben, können Ressourcen auch über diese Attribute gefunden werden. Zudem besitzt jeder Eintrag einer Ressource eine Lebenszeit, so dass der Eintrag nach Ablauf der Lebenszeit entfernt wird, sofern er vorher nicht aufgefrischt wird. Außerdem ist es möglich den Ressourcen Namensräume zuzuordnen. Über CoAP RDs können andere Geräte somit effizienter die von den Geräten der Umgebung angebotenen Ressourcen ermitteln, da die Informationen über die angebotenen Ressourcen nicht von jedem Gerät separat abgerufen werden müssen. Für Geräte, die zum Beispiel aufgrund von Batteriebetrieb nicht ununterbrochen mit dem Netzwerk verbunden sein können, bietet dies außerdem die Möglichkeit, die eigenen Ressourcen auch dann für andere Geräte auffindbar zu machen, wenn sie derzeit über keine aktive Netzwerkverbindung verfügen. Diese Methode wurde nicht für das Konzept dieser Arbeit verwendet, da es sich derzeit noch in der Entwicklung befindet.

Caturano et al. [CJR19] beschreiben ein Konzept zur automatisierten Erkennung von IoT-Geräten, das verschiedene von CoAP bereitgestellte Mechanismen zur Geräteerkennung kombiniert. Die einfachste Methode basiert darauf, dass es zwischen einigen Geräten bereits erstellte Relationen gibt, die traversiert werden können, um Informationen über weitere Geräte zu finden. Die zweite Methode setzt darauf via CoAP-Multicast-Discovery weitere Geräte zu finden, und damit wie bei der ersten Methode vorzugehen. Die dritte Methode verwendet das CoAP RD. Wird bei der Anwendung der ersten beiden Methoden ein CoAP RD gefunden, so registriert sich das Gerät dort und verwendet es, um weitere Geräte zu finden. Da die Geräte im Konzept dieser Arbeit anfangs über keine vordefinierten Relationen zu anderen Geräten verfügen, funktioniert die erste Methode alleine nicht. Methode eins und zwei würden zusammen funktionieren, bringen aber im gewünschten Anwendungsfall keinen Mehrwert gegenüber der reinen CoAP-Multicast-Discovery. Die dritte Methode hätte Vorteile gegenüber der reinen Multicast-Discovery, verwendet allerdings das noch nicht fertig entwickelte CoAP RD.

Yachir et al. [YDZ+17] beschreiben eine Erweiterung von CoAP und dem CoAP RD, um die Beschreibung von Ressourcen mit semantischen Informationen zu ergänzen, damit Ressourcen anhand von semantischen Anforderungen gefunden werden können. Dazu definieren sie ein universelles semantisches Modell, das ein Gerät und die Entitäten in seinem Umfeld beschreibt. Dieses Modell enthält Informationen über den Ort des Geräts, über Beziehungen zu Personen, über Art bzw. Bauweise des Geräts (Appliance) und Informationen über die Dienste des Geräts. So kann mithilfe des Modells beispielsweise herausgefunden werden, ob es sich bei einem Gerät um ein tragbares Gerät (Wearable) handelt und wer es trägt. Anschließend werden CoAP und das CoAP RD so erweitert, dass Ressourcen zusammen mit diesen semantischen Informationen im CoAP RD registriert und gesucht werden können. Das CoAP RD erhält die Fähigkeit gesuchte

Eigenschaften mit den Eigenschaften der registrierten Ressourcen zu vergleichen und anhand der Übereinstimmung zu bewerten. So können die Ressourcen ermittelt werden, die die Anforderungen am besten erfüllen.

Universal Plug and Play (UPnP) [DRB+15] ist ein Protokoll zur herstellerübergreifenden Ansteuerung von Geräten. Dazu besitzt UPnP ebenfalls Mechanismen, um Geräte im Netzwerk zu finden und eine maschinenlesbare Beschreibung dieser Geräte abzurufen. Zum Finden der Geräte verwendet UPnP das Simple Service Discovery Protocol (SSDP), welches ebenfalls in der UPnP Spezifikation definiert wird. SSDP verwendet, wie auch die CoAP-Multicast-Service-Discovery, Multicast-Kommunikation. Um sich den anderen Teilnehmern bekannt zu machen, sendet ein Gerät periodisch Nachrichten an eine Multicast-Gruppe. Außerdem können Geräte Suchanfragen verwenden, um nach anderen Teilnehmern mit bestimmten Eigenschaften zu suchen. SSDP könnte für die Erkennung von Geräten für das Konzept dieser Arbeit verwendet werden, erzeugt allerdings durch die ständige Multicast-Kommunikation im Vergleich zur CoAP-Multicast-Service-Discovery eine größere Belastung für das Netzwerk.

Breitbach et al. [BSEB19] beschreiben in ihrer Publikation mehrere Verfahren zur Verarbeitung und Auslagerung von Daten in Edge-Computing-Umgebungen, was für IoT-Geräte verwendet werden kann. Ziel dieser Verfahren ist es, die Latenz bei der Aufgabenausführung zu minimieren. Dazu entkoppeln sie die Daten, die für die Aufgabenausführung benötigt werden, von den Aufgaben. Um die Daten und Aufgaben auf die Knoten zu verteilen, setzen sie einen kontextsensitiven Multilevel Scheduler ein. Für diesen Scheduler stellen sie verschiedene Algorithmen zur Datenplatzierung, Aufgabenverteilung und Laufzeitanpassung vor, die sie anschließend evaluieren. Um die Aufgabenausführung zu beschleunigen, verwenden die Algorithmen zur Datenplatzierung verschiedene Strategien, um die Daten bereits präemptiv auf die Knoten zu übertragen. Diese Strategien werden in eine einfache Replikation und eine kontextsensitive Replikation unterteilt. Für die Algorithmen der Aufgabenverteilung werden die folgenden Strategien vorgestellt: Zufällige Verteilung, Verteilung anhand der Datenverfügbarkeit und Verteilung anhand der Leistung der Knoten. Bei der zufälligen Verteilung werden die Aufgaben wahllos auf Knoten verteilt, ungeachtet dessen, ob auf den Knoten die benötigten Daten verfügbar sind. Bei der Verteilung anhand der Datenverfügbarkeit wird ein Knoten zufällig ausgewählt, der die benötigten Daten besitzt. Bei der Verteilung anhand der Leistung der Knoten werden die Aufgaben an die Knoten zugewiesen, die die benötigten Daten und die meiste Leistung besitzen. Die Fluktuation in der Verfügbarkeit von Knoten hat Auswirkungen auf die Anzahl der verfügbaren Daten-Replikate. Um diese Schwankungen im Betrieb auszugleichen, verwendet der Scheduler unterschiedliche Strategien: statische Laufzeitanpassung, dynamische Laufzeitanpassung oder Laufzeitanpassung mithilfe von Daten-Caching. Bei der statischen Laufzeitanpassung wird die initial festgelegte Anzahl an Replikaten wiederhergestellt. Bei der dynamischen Laufzeitanpassung werden Kontextinformationen verwendet, um die geeignete Anzahl der Replikationen zu bestimmen. Bei der Laufzeitanpassung mithilfe von Daten-Caching speichert jeder Knoten eingehende Daten in einem Cache, um diese später wieder zu verwenden. Die Evaluation der Algorithmen hat gezeigt, dass die kontextsensitive Replikationsstrategie in Kombination mit der Aufgabenverteilung anhand der Leistung der Knoten und der dynamischen Laufzeitanpassung die effektivste Lösung darstellt. Da diese Architektur den Fokus auf die Platzierung von Daten legt, lässt sie sich nicht für das Konzept dieser Bachelorarbeit anwenden, da der Fokus dort primär auf der Platzierung von Operatoren anhand der Datenflussmodelle liegt.



---

Moderne Smartphones besitzen viele Sensoren und sind mit dem Internet verbunden. Sheng et al. stellen in [STXX13] ein Konzept vor, um diese Sensordaten von Smartphones über das Internet für verschiedene Anwendungsbereiche verfügbar zu machen: Sensing-as-a-Service (S<sup>2</sup>aaS). Die Sensordaten können beispielsweise für Anwendungen im Gesundheitswesen, Transportwesen oder zur Umweltbeobachtung verwendet werden. Den Kern des Konzepts stellen sogenannte Sensing-Server dar. Diese koordinieren den Austausch der Sensordaten. Um Sensordaten zur Verfügung zu stellen, müssen die Smartphone-Nutzer auf ihren Geräten eine App installieren. Diese App verbindet das Smartphone mit dem Sensing-Server. Um Sensordaten zu erhalten wird eine Anfrage an den Sensing-Server gesendet, in der die benötigten Sensordaten spezifiziert sind. Der Sensing-Server ermittelt daraufhin Smartphones, welche die geforderten Sensordaten liefern können und ruft die Daten von diesen ab. Zudem befassen sich Sheng et al. mit der Herausforderung energieeffiziente Algorithmen zu finden. Außerdem gehen sie in ihrer Arbeit auf Methoden ein, um Smartphone-Nutzern Anreize zu schaffen, die Sensoren ihrer Geräte für S<sup>2</sup>aaS zur Verfügung zu stellen. Das Ziel dieser Arbeit besteht allerdings nicht darin, Sensordaten aus einem weiten Spektrum zu erhalten, sondern definierte Datenflussmodelle in der eigenen Umgebung auszuführen. Zudem ist es nicht das Ziel dieser Arbeit Sensordaten für andere Anwendungen über das Internet zur Verfügung zu stellen, so dass sich S<sup>2</sup>aaS nicht für die Verwendung in dieser Arbeit eignet.



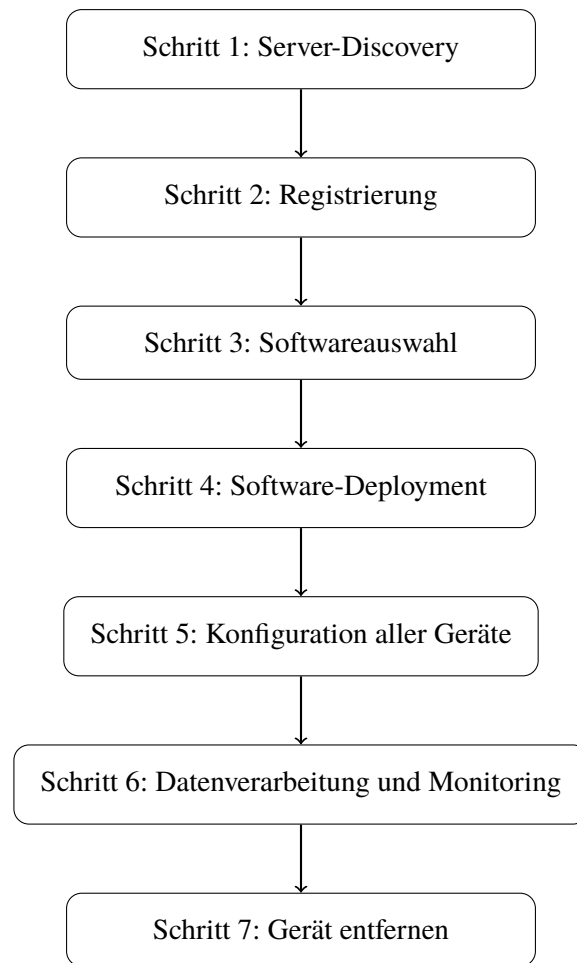
## 4 Konzept

In diesem Kapitel wird das Konzept meiner Arbeit, das sogenannte Constraint Device Management (CDM), vorgestellt. Das CDM umfasst eine Methode und eine Plattform, die es ermöglichen, neue Geräte in einer bestehenden Umgebung automatisiert zu erkennen, diese in die Umgebung einzubeziehen und entsprechend zu reagieren, wenn Geräte die Umgebung verlassen haben. Die manuelle Vorkonfiguration der Geräte ist dabei unabhängig von der spezifischen Umgebung in der das Gerät später eingesetzt wird. In Abschnitt 4.1 wird zuerst ein Überblick über das Konzept dieser Arbeit gegeben. In den Abschnitten 4.2 bis 4.8 werden schrittweise die Abläufe des Konzepts erläutert. In Abschnitt 4.9 wird auf verschiedene Ideen eingegangen, mit welchen das Konzept weiter optimiert werden kann.

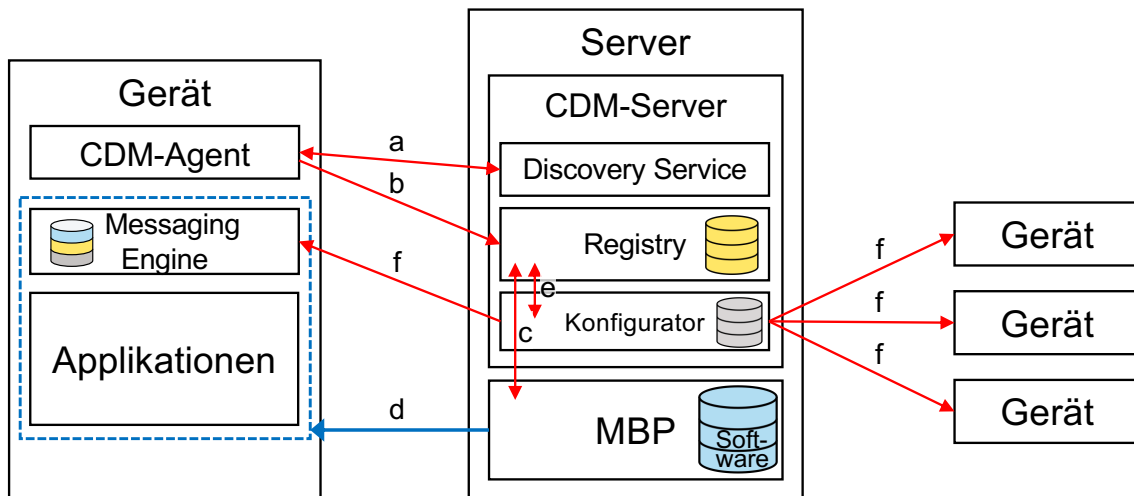
### 4.1 Überblick

Zum Einbeziehen neuer Geräte wird eine zentrale Server-Komponente, der sogenannte CDM-Server, verwendet, der dazu dient neue Geräte einzurichten und zu konfigurieren. Dieser ist nicht für die Kommunikation zwischen den einzelnen Geräten im Betrieb nicht relevant, da die Datenverbreitung auf den Geräten dezentral erfolgt. Die zentrale Server-Komponente wird eingesetzt, da eine ideale, vollständig dezentrale Lösung nur schwierig umzusetzen bzw. unter großen Einschränkungen möglich ist. Da die von einem Gerät benötigte Software erst beim Beitritt des Geräts auf diesem installiert werden kann, müssten die Geräte selbst als Software-Repository fungieren, um die Software für neue Geräte bereitzustellen. Zudem müsste sichergestellt werden, dass keine Softwareartefakte verloren gehen, wenn Geräte die Umgebung unerwartet verlassen. Da es sich bei diesen IoT-Geräten aber typischerweise um Geräte mit eingeschränkten Ressourcen, wie einem geringem Speicher handelt, ist es nicht praktikabel, diese sowieso geringen Ressourcen für das Bereitstellen von Software für andere Geräte zu verwenden. Deshalb werden für das Bereitstellen und Einrichten der Geräte sowie für das Bereitstellen der notwendigen Software zentrale Komponenten verwendet.

Sind die IoT-Geräte vollständig eingerichtet, erfolgt der Datenaustausch im Betrieb direkt zwischen den IoT-Geräten. Wenn sich die Konfiguration der Umgebung ändert, werden zentrale Komponenten erneut zur Einrichtung der Geräte benötigt. Das Konzept verwendet eine klassische Client-Server-Architektur. Dabei ist die Client-Applikation auf jedem IoT-Gerät installiert und wird CDM-Agent genannt. Außerdem besitzt die Umgebung eine Instanz des CDM-Servers sowie eine Instanz der Multi-purpose Binding and Provisioning Platform (MBP). Der CDM-Server verfügt über Datenbanken zum Speichern von Datenflussmodellen und Geräteinformationen; außerdem eine weitere Datenbank, die als Zwischenspeicher für Daten der Messaging Engine (ME) verwendet wird. Ebenfalls besitzt der CDM-Server entsprechende Schnittstellen, um die registrierten Geräte und gespeicherten Datenflussmodelle zu verwalten. Dabei handelt es sich um Datenflussmodelle entsprechend der Definition aus dem Grundlagenkapitel zur ME (Abschnitt 2.2). Abbildung 4.1 zeigt den Ablauf der Methode, um ein neues Gerät in die Umgebung zu integrieren. Diese Methode

**Abbildung 4.1:** Konzeptübersicht

ist dabei in sieben Schritte unterteilt: Im ersten Schritt versucht der CDM-Agent zunächst mittels Multicast-Nachrichten den CDM-Server in der Umgebung zu finden (Abbildung 4.2, a). Sollte das Netzwerk der Umgebung keine Multicast-Kommunikation unterstützen, können alternative Methoden eingesetzt werden, um den Server zu ermitteln. Dies wird in Abschnitt 4.2 behandelt. Im zweiten Schritt sendet der CDM-Agent eine Registrierungsanfrage mit Informationen über das Gerät an den CDM-Server, wodurch die Registrierung angestoßen wird (Abbildung 4.2, b). Im dritten und vierten Schritt bestimmt der CDM-Server die Software, die auf dem Gerät installiert werden soll und installiert diese auch zusammen mit der ME mithilfe der MBP auf dem Gerät (Abbildung 4.2, c und d). Im fünften Schritt wird die ME auf dem neuen Gerät konfiguriert und die MEs auf allen anderen Geräten erhalten Informationen über das neue Gerät (Abbildung 4.2, e und f). Im sechsten Schritt ist das Gerät vollständig in der Umgebung integriert und kann Daten verarbeiten. Außerdem wird die Anwesenheit des Gerätes überwacht. Schritt sieben tritt ein, wenn das Gerät die Umgebung verlässt. Dabei wird unterschieden, ob das Gerät erwartet oder unerwartet aus der Umgebung entfernt wurde. In beiden Fällen wird es aus der ME-Konfiguration der anderen Geräte entfernt. Beim erwarteten Entfernen wird der CDM-Server darüber informiert, dass das Gerät die Umgebung bald verlassen wird. In diesem Fall können unverarbeitete Daten von dem Gerät extrahiert werden. Diese Daten werden bevorzugter Weise direkt auf andere kompatible Geräte zur



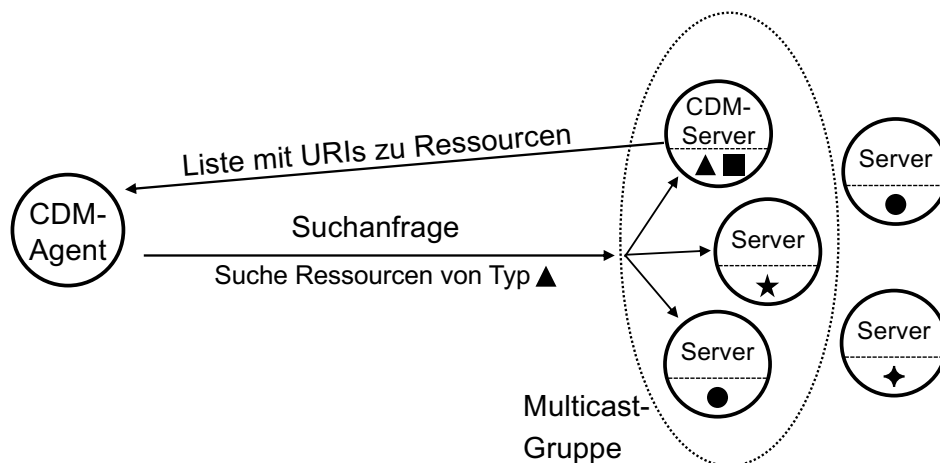
**Abbildung 4.2:** Konzeptionelle Darstellung der Registrierung eines neuen Geräts (links) und der daraus resultierenden Konfiguration bestehender Geräte (rechts)

Weiterverarbeitung übertragen. Für Daten, für die keine kompatiblen Geräte zur Weiterverarbeitung verfügbar sind, wird alternativ eine Datenbank auf dem Server zur Zwischenspeicherung verwendet. Betritt ein Gerät den sechsten Schritt wird geprüft, ob das Gerät in der Lage ist, Daten aus dieser Datenbank zu verarbeiten. Ist dies der Fall, werden diese Daten auf das Gerät zur Verarbeitung übertragen und aus der Datenbank entfernt. Ist das Gerät nicht in der Lage alle Daten aus der Datenbank zu verarbeiten, so verbleiben die restlichen Daten weiterhin in der Datenbank und das Prozedere wird erneut gestartet, wenn ein weiteres Gerät den sechsten Schritt betritt. Wird der Server hingegen vor dem Entfernen des Geräts nicht darüber informiert, beispielsweise wenn das Gerät ausfällt, handelt es sich um das unerwartete Entfernen. Hierbei können keine Daten von dem Gerät extrahiert werden.

Die Registrierung von neuen Geräten kann jederzeit angestoßen werden. Ebenfalls können über die Datenflussschnittstelle des CDM-Servers zu jeder Zeit weitere Datenflussmodelle der Datenflussmodell-Datenbank hinzugefügt oder entfernt werden. Geschieht dies, werden diese neuen Datenflussmodelle, unabhängig von der beschriebenen Methode zum Hinzufügen neuer Geräte, auf alle registrierten Geräte übertragen. Analog wird beim Entfernen bestehender Datenflussmodelle verfahren. In den folgenden Abschnitten werden die einzelnen Schritte der in Abbildung 4.2 dargestellten Methoden erläutert.

## 4.2 Schritt 1: Server-Discovery

Das Gerät soll in verschiedenen Umgebungen funktionieren. Deshalb besitzt es in seinem Initialzustand noch keine Informationen über eine spezifische Umgebung. So kennt das Gerät zu Beginn nicht die Adresse des CDM-Servers, da diese in jeder Umgebung unterschiedlich sein kann. Deshalb ist es notwendig, dass der CDM-Agent die Adresse des CDM-Servers für die derzeitige Umgebung herausfindet. Dazu werden in den folgenden Abschnitten drei Methoden beschrieben, wobei die erste Methode genauso von CoAP verwendet wird. Bevorzugt wird die CoAP-Funktion zur Service-Discovery in Kombination mit Multicast-Kommunikation eingesetzt.



**Abbildung 4.3:** Grafische Darstellung der Multicast-Discovery

Diese Kommunikation ist in Abbildung 4.2 a dargestellt. Sollte in dem Netzwerk der Umgebung keine Multicast-Kommunikation möglich sein, wird alternativ DHCP oder DNS verwendet, um die Adresse des CDM-Servers zu bestimmen.

#### 4.2.1 Multicast-Discovery

Mit Broadcasts können Nachrichten an mehrere Empfänger versendet werden, ohne dass der Sender explizit die Netzwerkadresse der Empfänger kennen muss. Durch diese Eigenschaft können Broadcasts in einem Protokoll verwendet werden, um Teilnehmer eines Netzwerks zu finden. Da aber unter Umständen nicht alle Netzwerkteilnehmer ein Interesse daran haben, solch ein Protokoll zu verwenden, ist es sinnvoller, Multicast-Kommunikation für solch ein Protokoll zu verwenden. Ein naiver Ansatz für solch ein Protokoll wäre der Folgende: Ein Netzwerkteilnehmer sucht nach einem anderen Teilnehmer mit bestimmten Eigenschaften. Deshalb sendet er eine Nachricht an eine Multicast-Gruppe. Die Teilnehmer dieser Gruppe antworten ihm daraufhin mit Informationen über sich. Er analysiert nun die erhaltenen Informationen und prüft, ob ein Teilnehmer die von ihm gesuchten Eigenschaften erfüllt. Die Antworten werden nicht an die Multicast-Gruppe gesendet, da diese Nachricht dann an alle Teilnehmer übermittelt werden müssten. Stattdessen werden die Antworten als Unicast-Nachricht direkt an den Absender der Anfrage gesendet. Dieser Ansatz wäre für das Konzept dieser Arbeit ausreichend, kann allerdings noch verbessert werden. Um die Anzahl der benötigten Nachrichten zu reduzieren, werden die gesuchten Eigenschaften bereits in die Suchanfrage mit aufgenommen. So kann jeder Netzwerkteilnehmer, der diese Anfrage erhält, selbst überprüfen, ob er die gesuchten Eigenschaften besitzt. Nur wenn er diese besitzt, sendet er auch eine Antwort an den Absender. Besitzt er die gesuchten Eigenschaften nicht, dann sendet er keine Informationen über sich, da diese für den Absender der Anfrage nicht von Nutzen sind. Auf diese Weise werden nur Informationen übertragen, deren Übertragung tatsächlich notwendig ist. Dabei wird die Anzahl der zu sendenden Nachrichten minimiert, was auch die Belastung für das Netzwerk und die anderen Teilnehmer gering hält.

Das verbesserte Protokoll ist in Abbildung 4.3 dargestellt. Dabei stellen die Formen in der unteren Hälfte der Server-Kreise die verschiedenen Ressourcen dar, die von den Servern angeboten werden. Konkret sieht das Protokoll wie folgt aus: Viele Server sind Teilnehmer einer Multicast-Gruppe

und bieten verschiedene Dienste an, darunter auch der CDM-Server. Der CDM-Agent möchte einen Dienst (in Abbildung 4.3 als ▲-Symbol dargestellt) nutzen, der vom CDM-Server angeboten wird. Der CDM-Agent kennt aber die Netzwerkadresse des CDM-Servers nicht. Der CDM-Agent sendet ein UDP Paket an die Adresse der Multicast-Gruppe des Servers. Das Paket enthält bereits Informationen darüber, nach welchem Dienst der CDM-Agent sucht. Die Server empfangen das Paket und prüfen, ob sie den gesuchten Dienst anbieten. Der CDM-Server bietet den gesuchten Dienst an und sendet eine Antwort direkt an den CDM-Agenten. Die Antwort enthält Informationen über den Dienst, die der CDM-Agent benötigt, um den Dienst zu nutzen. Die anderen Server, die den gesuchten Dienst nicht anbieten, antworten nicht auf die Anfrage. In diesem Fall ist eine Antwort nicht sinnvoll, da die Information über die Nichtexistenz des Dienstes auf einem der Server für den CDM-Agent nicht von Nutzen ist. Damit der CDM-Agent dieses Prozedere nicht jedes Mal durchführen muss, wenn er den Dienst des CDM-Servers nutzen möchte, speichert er die erhaltenen Informationen in einem Zwischenspeicher, dem sogenannten Cache. Jedes weitere Mal, wenn der CDM-Agent den Dienst des CDM-Servers nutzen möchte, kann er zuerst versuchen, den Server mit den im Cache gespeicherten Informationen zu erreichen. Verwendet der CDM-Agent die Informationen aus dem Cache und es treten Verbindungsfehler bei der Nutzung des Dienstes auf, deutet dies darauf hin, dass die im Cache gespeicherten Informationen obsolet sind. Dies kann zum Beispiel auftreten, wenn sich die Netzwerkadresse des CDM-Servers geändert hat. In diesem Fall muss der CDM-Agent das Prozedere erneut durchführen. Wenn sich aber an der Netzwerkverbindung zwischen CDM-Agent und Server nichts ändert, sollte dieser Fall normalerweise nicht auftreten. So kann mithilfe des Cache die Anzahl der im Betrieb gesendeten Multicast-Nachrichten sehr gering gehalten werden. Die dadurch verursachte Belastung für das Netzwerk und dessen Teilnehmer wird minimiert.

Erhält der CDM-Agent auf Anfragen an die Multicast-Gruppe keine Antwort, ist der CDM-Server entweder nicht erreichbar oder das Netzwerk erlaubt keine Multicast-Kommunikation. Ist kein CDM-Server im Netzwerk verfügbar, dann kann der CDM-Agent nichts daran ändern. Erlaubt das Netzwerk allerdings lediglich keine Multicast-Kommunikation, dann kann der CDM-Agent über andere Wege versuchen, die nötigen Informationen über den CDM-Server zu erhalten. In großen Netzwerken mit sehr vielen Teilnehmern werden Broad- und Multicast-Nachrichten oft nur eingeschränkt weitergeleitet, da diese zu Überlastungen der Netzwerke führen können. In diesem Fall kann eines der beiden Verfahren verwendet werden, die im Folgenden beschrieben werden.

### 4.2.2 Discovery mit DHCP

Jedes Gerät in einem IP-Netzwerk benötigt eine IP-Adresse, wobei jede Adresse eindeutig zu einem Gerät gehören muss. Wird dieselbe IP-Adresse von mehreren Geräten im selben Netzwerk verwendet, ist die Kommunikation dieser Geräte gestört. Damit ein Gerät in einem IP-Netzwerk uneingeschränkt kommunizieren kann, müssen neben der IP-Adresse noch weitere Parameter korrekt konfiguriert sein, die vom jeweiligen Netzwerk abhängig sind. Diese Konfiguration wird als Netzwerkkonfiguration bezeichnet. Damit diese Netzwerkkonfiguration nicht manuell vorgenommen werden muss, wenn ein Gerät ein Netzwerk betritt, wird in der Regel das Dynamic Host Configuration Protocol (DHCP) [Dro97] eingesetzt. DHCP sorgt dafür, dass ein Gerät in einem Netzwerk automatisch eine IP-Adresse und die weitere Netzwerkkonfiguration erhält. Dazu besitzt das Netzwerk einen DHCP-Server. Dieser hat die Aufgabe, den Geräten IP-Adressen zuzuteilen, den Überblick über die vergebenen IP-Adressen zu behalten und den Geräten die weitere Netzwerkkonfiguration mitzuteilen. Benötigt ein Gerät eine Adresse, sendet es einen Broadcast in das Netzwerk, worauf der DHCP-Server

antwortet. Diese Antwort kann dabei entweder per Broadcast oder Unicast übermittelt werden. In Netzwerken, die keine Broadcasts erlauben, werden diese Anfragen nicht von den Routern und Switches weitergeleitet, die die Netzwerksegmente miteinander verbinden. Wird in diesen Netzwerken dennoch DHCP verwendet, kommen DHCP Relay Agents [Pat01] auf diesen Routern und Switches zum Einsatz. Empfängt der Relay Agent einen DHCP-Broadcast von einem Gerät, leitet er die Anfrage via Unicast über das innere Netzwerk an den DHCP-Server weiter. Der Relay Agent erhält die Antwort des DHCP-Servers ebenfalls über Unicast und sendet sie als Broadcast in das Netzwerksegment des anfragenden Geräts oder mit Unicast direkt an das anfragende Gerät. DHCP bietet auch die Möglichkeit eigene Erweiterungen zu entwickeln, die es ermöglichen, anwendungsspezifische Daten über DHCP zwischen DHCP-Server und Geräten auszutauschen. So wäre es prinzipiell möglich die Adresse des CDM-Servers über DHCP an die Geräte zu übertragen. Dazu müsste allerdings die Implementierung eines DHCP-Server entsprechend erweitert werden und genau dieser DHCP-Server in allen Umgebungen eingesetzt werden, in denen dieses Konzept verwendet werden soll. Der DHCP-Server in einem Netzwerk ist aber typischerweise fest in anderen Systemen, wie zum Beispiel einem Router, integriert, in denen die Implementierung nicht einfach ausgetauscht werden kann. Deshalb ist diese Lösung nicht praktikabel.

### 4.2.3 Discovery mit DNS

Im Internet werden URLs mit Domainnamen verwendet, um zum Beispiel Internetseiten aufzurufen. Diese Namen sind leichter zu merken als die IP-Adressen der Server. Damit der Computer sich mit dem gewünschten Server verbinden kann, benötigt er allerdings die IP-Adresse des Servers. Deshalb wird das Domain Name System (DNS) [Moc87] verwendet, um Domainnamen zu den jeweiligen Serveradressen aufzulösen. Vereinfacht könnte DNS mit einem Telefonbuch für das Internet verglichen werden. Will ein Computer eine Domain auflösen, übermittelt er den Domainnamen an einen DNS-Server und fragt nach der dazugehörigen IP-Adresse. Damit dies möglich ist, muss der Computer allerdings die IP-Adresse eines DNS-Servers kennen. Wird diese Adresse nicht manuell konfiguriert, erhält er diese in der Regel im Rahmen seiner Netzwerkkonfiguration über DHCP. Wie bereits beschrieben kann DHCP auch in Netzwerken ohne Broadcast-Kommunikation eingesetzt werden. So könnte DNS für dieses Konzept eingesetzt werden, um die Netzwerkadresse des CDM-Servers in der Umgebung zu ermitteln. Dabei besitzt die Umgebung einen DNS-Server auf dem die Adresse des CDM-Servers der Umgebung unter einem festgelegten Domainnamen hinterlegt ist.

## 4.3 Schritt 2: Registrierung

Nachdem der CDM-Agent auf dem Gerät die Netzwerkadresse des CDM-Servers in Erfahrung gebracht hat, kann er sich jetzt an diesem registrieren. Da der CDM-Agent nicht weiß, ob das Gerät in der Vergangenheit bereits registriert wurde, sendet er allerdings erst eine Heartbeat-Nachricht (siehe Abschnitt 4.7.1) an den CDM-Server. Da das Gerät noch nicht registriert ist, lehnt der CDM-Server die Anfrage ab und der CDM-Agent weiß, dass eine Registrierung notwendig ist. Zur Registrierung sendet der CDM-Agent eine Registrierungsanfrage mit Informationen über das Gerät an den CDM-Server. Diese Übertragung ist in Abbildung 4.2 durch Pfeil b dargestellt. Diese Anfrage beinhaltet unter anderem:



**Eindeutige ID** wird benötigt, um ein Gerät eindeutig identifizieren zu können.

**Netzwerkadresse des Geräts** wird benötigt, um auf das Gerät zwecks Softwareinstallation und Konfiguration zuzugreifen. Außerdem wird die Netzwerkadresse von anderen Geräten benötigt, um dieses Gerät zu kontaktieren.

**Gültigkeitsdauer der Anfrage** dient dazu, dass unerwartete Entfernungen bzw. den Ausfall von Geräten zu erkennen. Innerhalb der Gültigkeitsdauer der Anfrage muss das Gerät eine weitere Anfrage an den CDM-Server senden, um die Gültigkeit der Registrierung aufzufrischen. Erhält der CDM-Server in diesem Zeitraum keine erneute Anfrage, wird davon ausgegangen, dass das Gerät die Umgebung verlassen hat und die Registrierung wird aufgehoben. Die Gültigkeitsdauer in der Anfrage wird in Sekunden angegeben und daraus erst auf dem CDM-Server der absolute Zeitpunkt berechnet, bis zu dem die Anfrage gültig ist. Dadurch entstehen keine Probleme, falls die Uhrzeit bzw. das Datum auf dem Gerät nicht korrekt eingestellt ist. Die Zeit der Gültigkeitsdauer einer Anfrage kann entweder statisch, immer gleich, oder während der Laufzeit dynamisch, nach Bedarf, bestimmt werden.

**Fähigkeiten des Geräts** werden im nächsten Schritt dazu verwendet, um zu diesem Gerät passende Operationen zu bestimmen.

**Name der Schlüsselgruppe des Geräts** wird verwendet, um die Zugangsdaten für das Gerät zu ermitteln, die bei der Installation der Software auf das Gerät benötigt werden. Eine Schlüsselgruppe ist dazu an, auf dem CDM-Server gespeicherte, Zugangsdaten gebunden. Dadurch können dieselben Zugangsdaten für verschiedene Geräte verwendet werden.

Wenn der CDM-Server eine Anfrage erhält, berechnet er den absoluten Zeitpunkt bis zu dem die Anfrage gültig ist. Anschließend werden die erhaltenen Informationen in der Gerätedatenbank gespeichert und im nächsten Schritt die Software bestimmt, die auf das Gerät installiert werden soll.

## 4.4 Schritt 3: Softwareauswahl

In diesem Schritt werden die Operationen bestimmt, die auf dem Gerät ausgeführt werden können und damit auch die Software, die auf dem Gerät installiert werden muss. Die Auswahl erfolgt in mehreren Schritten. Zunächst werden die Anforderungen der verfügbaren Operationen und die Fähigkeiten des Geräts dazu verwendet, um die zum Gerät passenden Operationen zu ermitteln. Da die Installation von Software für jede Operation Speicherplatz auf dem Gerät verbraucht, kann unter Umständen nicht für jede passende Operation die Software auf ein einziges Gerät installiert werden. Deshalb wird jeder passenden Operation eine Gewichtung gegeben, durch die bestimmt wird, welche Operationen bevorzugt installiert werden sollen. Passende Operationen können der Nutzung von spezieller Hardware des Geräts dienen, wie dem Zugriff auf angeschlossene Sensoren oder Aktoren. Da die Nutzung dieser Sensoren und Aktoren essenziell ist, werden die dafür notwendigen passenden Operationen besonders behandelt. Für die anderen Operatoren werden anschließend Informationen über die Auslastung der Umgebung ermittelt, welche zur Berechnung der Gewichte verwendet werden. Aus der Datenbank mit den Datenflussmodellen der Umgebung wird die Menge

der verfügbaren Operationen  $O$  der Umgebung ermittelt. Jede Operation  $o \in O$  besitzt eine Menge von Anforderungen

$$A(o) = \{a_0, \dots, a_n\} \quad (4.1)$$

Damit eine Operation für die Ausführung auf einem Gerät in Frage kommt, muss das Gerät über entsprechende Fähigkeiten verfügen. Ein Gerät  $d \in D$  besitzt dazu eine Menge

$$F(d) = \{f_0, \dots, f_n\} \quad (4.2)$$

mit den Fähigkeiten des Geräts. Die Funktion

$$G: A \times F \rightarrow \{true, false\} \quad (4.3)$$

gibt Auskunft darüber, ob eine Anforderung einer Operation von einer Fähigkeit eines Geräts erfüllt wird. Damit eine Operation  $o$  zu einem Gerät  $d$  passt, muss das Gerät für jede Anforderung  $a \in A(o)$  der Operation eine Fähigkeit  $f \in F(d)$  besitzen, sodass  $G$  für dieses Paar den Wert *true* liefert. Diese Bedingung wird formalisiert in der Funktion

$$P(d) = \{o \in O \mid \forall a \in A(o) \exists f \in F(d) : G(a, f) = true\} \quad (4.4)$$

verwendet.  $P(d)$  beschreibt damit die Menge der Operationen, die zu einem Gerät  $d$  passen. Nun wird für diese passenden Operationen der Bedarf in der Umgebung bestimmt. Dazu wird zum einen mithilfe der Gerätedatenbank bestimmt, wie oft jede der passenden Operationen bereits in der Umgebung von einem Gerät ausgeführt wird. Anschließend wird die Auslastung für diese Operationen auf den Geräten geprüft.

$$S_O: O \rightarrow \mathbb{N}_0 \quad (4.5)$$

gibt den Speicher in Bytes an, den eine Operation auf einem Gerät voraussetzt und

$$S_D: D \rightarrow \mathbb{N}_0 \quad (4.6)$$

gibt den auf einem Gerät für Operatoren verfügbaren Speicher in Bytes an. Sei

$$I(d) \subseteq O \quad (4.7)$$

die Menge der Operationen deren Software auf ein Gerät  $d$  installiert werden soll. Dann gilt für  $I(d)$  die Bedingung

$$\sum_{i \in I(d)} S_O(i) \leq S_D(d) \quad (4.8)$$

Da die ME ausgehende Daten entsprechend der Last auf die verfügbaren Geräte verteilt, wird die Last pro Gerät reduziert, wenn ein weiteres Gerät für die ausgelastete Operation verfügbar wird. Deshalb ist es sinnvoll, Operationen auf neuen Geräten verfügbar zu machen, wenn diese auf den derzeitigen Geräten der Umgebung ausgelastet sind. Des Weiteren ist es aber auch sinnvoll, jede Operation auf einem bestimmten Anteil der vorhandenen Geräte verfügbar zu machen, damit der Betrieb der Umgebung bei Aus- oder Wegfall von Geräten gewährleistet bleibt. Diese beiden Faktoren sollten bei der Bestimmung der Gewichte berücksichtigt werden. Sei also

$$L: O \rightarrow [0, 1] \quad (4.9)$$

die derzeitige prozentuale Auslastung einer Operation in der Umgebung. Für die Art von Ressourcen, die diese Operation benötigt, wird dazu auf allen Geräten, die diese anbieten, die angebotenen und verwendeten Ressourcen jeweils aufsummiert und das Verhältnis bestimmt. Der arithmetische Mittelwert der prozentualen Auslastung der einzelnen Geräte wäre nicht repräsentativ für die Gesamtauslastung, da sich die angebotenen Ressourcen der einzelnen Geräte stark unterscheiden können. Weiter sei

$$C: O \rightarrow \mathbb{N}_0 \quad (4.10)$$

die Anzahl der Geräte in der Umgebung auf denen die Operation verfügbar ist.

$$H(d) \subseteq P(d) \quad (4.11)$$

sei die Menge der Operationen für die Nutzung spezieller Hardware des Geräts. So sei

$$w^D(o) = \begin{cases} 1, & C(o) = 0 \\ (1 - \alpha) \cdot L(o) + \alpha \cdot (1 - \frac{C(o)}{|D|}), & \text{sonst} \end{cases} \quad (4.12)$$

das Gewicht mit dem eine Operation auf das neue Gerät installiert werden soll. Dabei wird mit  $\alpha \in [0, 1]$  gewählt, ob die derzeitige Auslastung der Operation in der Umgebung oder das Verhältnis der für die Operation verfügbaren Geräte einen höheren Einfluss auf das Gewicht hat. Damit die Umgebung den Betrieb aufrechterhalten kann, muss jede Operation auf mindestens einem Gerät verfügbar sein. Deshalb erhält eine Operation ein Gewicht von 1, wenn sie auf keinem Gerät der Umgebung verfügbar ist. Für  $\alpha = 0$  wird nur die Last der Umgebung einbezogen, für  $\alpha = 1$  nur das Verhältnis der verfügbaren Geräte. So sorgt ein großer Wert für  $\alpha$  dafür, dass die Auslastung der Geräte geringer gehalten wird, was allerdings auf Kosten der Stabilität der Umgebung geht. Ist eine Operation im schlechtesten Fall nur auf einem einzigen Gerät verfügbar, kann der Ausfall dieses Geräts unter Umständen den Betrieb der Umgebung gefährden. Ein kleiner Wert für  $\alpha$  sorgt dafür, dass die Umgebung durch die höhere Redundanz stabiler gegenüber Ausfällen ist, was allerdings eine geringere Leistungsfähigkeit der Umgebung zur Folge haben kann.

Um die Menge an Operationen zu bestimmen, die letztendlich auf das Gerät installiert werden soll, wird ein Greedy-Algorithmus verwendet. Auf Eingabe eines Geräts bestimmt er die Menge der Operationen die auf dem Gerät installiert werden sollen. Er ist als Pseudocode in Algorithmus 4.1 dargestellt. Der Algorithmus fügt zuerst alle Operationen der Ausgabemenge hinzu, die für die Nutzung spezieller Hardware des Geräts benötigt werden. Dabei wird der danach verbleibende Gerätespeicher  $s$  bestimmt. Es wird davon ausgegangen, dass der Gerätespeicher für die Installation dieser Operationen ausreicht, da die Software für diese Operationen auf die Hardware des Geräts zugeschnitten ist. Wenn der Algorithmus dabei dennoch um eine Prüfung des Gerätespeichers erweitert werden sollte, dann müsste entschieden werden, welche dieser Operationen wichtiger sind als andere. Dazu wäre es allerdings notwendig die Semantik der Operationen auf Anwendungsebene interpretieren zu können, was außerhalb dem Fokus dieser Arbeit liegt. Für die restlichen passenden Operationen wird anschließend, in der Reihenfolge absteigend nach ihrem Gewicht  $w^D$  geprüft, ob der übrige Speicher für die Installation ausreicht. Ist der Speicher ausreichend, wird die Operation zur Ausgabemenge hinzugefügt und der verbleibende Speicher verringert.

**Algorithmus 4.1** Greedy-Algorithmus zur Bestimmung der Operationen für ein Gerät

---

```
function GREEDY( $d$ )
   $I \leftarrow \emptyset$  // Menge der ausgewählten Operationen
   $s \leftarrow S_D(d)$  // Verbleibender Gerätespeicher
  for all  $o \in H(d)$  do
     $I \leftarrow I \cup \{o\}$ 
     $s \leftarrow s - S_O(o)$  // Annahme: Gerätespeicher reicht für alle Operationen aus  $H(d)$  aus
  end for
   $T \leftarrow P(d) \setminus H(d)$ 
  while  $Z \neq \emptyset$  do
     $o \leftarrow$  Operation mit dem größten  $w^D(o)$  aus  $Z$ 
     $Z \leftarrow Z \setminus \{o\}$ 
    if  $s \geq S_O(o)$  then
       $I \leftarrow I \cup \{o\}$ 
       $s \leftarrow s - S_O(o)$ 
    end if
  end while
  return  $I$ 
end function
```

---

## 4.5 Schritt 4: Software-Deployment

In diesem Schritt wird die Software der zuvor ausgewählten Operationen zusammen mit der ME auf dem Gerät installiert und anschließend gestartet. Mithilfe der grafischen Benutzeroberfläche der MBP können Softwareartefakte verwaltet werden. Außerdem kann die MBP diese Softwareartefakte auf Geräten installieren, starten, stoppen und entfernen. Sie bietet eine REST-Schnittstelle an, über welche diese Aktionen von anderen Programmen gesteuert werden können. Aus diesem Grund wird die MBP in diesem Konzept zum Installieren der Software verwendet. Die Interaktion mit der MBP ist in Abbildung 4.6 durch Pfeil c dargestellt. Die Installation der Software ist Abbildung 4.6 durch Pfeil d dargestellt. Konzeptionell wäre die MBP allerdings auch durch eine andere Applikation ersetzbar, mit deren Hilfe Software auf die Geräte installiert werden kann. Vor der Softwareinstallation muss das Gerät zunächst an der MBP registriert werden. Da die MBP über SSH auf das Gerät zugreift, benötigt sie auch die entsprechenden Zugangsdaten, wie beispielsweise der private Schlüssel oder das Administratorpasswort für das Gerät. Diese Informationen erhält die MBP bei der Registrierung des Geräts von dem CDM-Server, so wie auch den Namen, die IP- und MAC-Adresse des Geräts.

MBP kategorisiert die Softwareartefakte in Sensoren und Aktoren, sodass auch Softwareartefakte als Sensor oder Aktor angelegt werden müssen, die sich tatsächlich in keine dieser Kategorien einteilen lassen. Dies stellt für das Konzept allerdings keine Einschränkung dar. Um die Software auf ein Gerät zu installieren, wird zuerst in der MBP eine Instanz der Software auf dem Gerät angelegt, die anschließend installiert und gestartet wird. Nach der Installation wird in der Gerätedatenbank vermerkt, welche Operationen erfolgreich auf dem Gerät installiert wurden.

## 4.6 Schritt 5: Konfiguration aller Geräte

Nach der Installation besitzt die ME keine Informationen über die Umgebung oder über das Gerät auf dem sie installiert ist. Um für die Umgebung Daten verarbeiten zu können, benötigt die ME folgende Informationen:

- i) Für jede in Schritt 4 installierte Software, jeweils den Installationsort und den Namen der dazugehörigen Operation
- ii) Die Datenflussmodelle, die zur Datenverarbeitung verwendet werden
- iii) Die Netzwerkadresse und die angebotenen Operationen von anderen Geräten in der Umgebung

Die Übertragung dieser Informationen ist in Abbildung 4.6 durch Pfeil f dargestellt. Die in i) beschriebenen Informationen sind nur von der installierten Software abhängig. Daher müssen diese immer dann erneut auf das Gerät übertragen werden, wenn weitere Operationen auf das Gerät installiert, oder bestehende Operationen entfernt werden. Ebenfalls können sich die in ii) und iii) beschriebenen Informationen im Betrieb jederzeit ändern, sodass diese auf dem Gerät aktualisiert werden müssen, wenn es Änderungen gibt. Die in ii) beschriebenen Informationen müssen auf den Geräten aktualisiert werden, wenn dem CDM-Server über die entsprechende Schnittstelle neue Datenflussmodelle hinzugefügt oder entfernt wurden. Die in iii) beschriebenen Informationen müssen immer dann auf den Geräten aktualisiert werden, wenn ein Gerät die Umgebung betreten oder verlassen hat.

Diese Anforderungen wären erfüllt, wenn der CDM-Server beim Hinzufügen eines Geräts alle anderen Geräte über das neue Gerät informieren würde und dem neuen Gerät die ihm bekannten, in i) - iii) beschriebenen Informationen zukommen lassen würde. Würde ein Gerät entfernt werden, müssten die Informationen über dieses nur von allen anderen Geräten wieder entfernt werden. Bei diesem naiven Ansatz verfügt jedes IoT-Gerät über alle dem CDM-Server bekannte Informationen. Da IoT-Geräte typischerweise nicht sehr leistungsstark sind und über begrenzten Speicher verfügen, ist dieser Ansatz nicht sehr skalierbar [GBMP13].

Im Allgemeinen benötigt nicht jedes Gerät alle Informationen über die gesamte Umgebung. In Bezug auf iii) muss jedes Gerät nur die anderen Geräte kennen, an die es tatsächlich Daten übermitteln muss. Aus den Datenflussmodellen lässt sich mit dem Namen einer Operation ermitteln, welche Namen die Operationen haben, die auf diese folgen können. Zur besseren Veranschaulichung wird in diesem Abschnitt folgendes Beispiel verwendet: In der Umgebung sind die Operationen  $o_1$  bis  $o_6$  definiert, wobei der Graph in Abbildung 4.4 das verwendete Datenflussmodell beschreibt. Der Name einer Operation  $o_i$  ist in diesem Beispiel ebenfalls  $o_i$ , sodass im Rahmen des Beispiels nicht zwischen einer Operation und dem Namen einer Operation unterschieden werden muss. Die Umgebung besitzt die drei Geräte  $d_1$ ,  $d_2$  und  $d_3$ , wobei auf Gerät  $d_1$  die Operationen  $o_1$  und  $o_6$ , auf Gerät  $d_2$  die Operationen  $o_2$  und  $o_4$  und auf Gerät  $d_3$  die Operationen  $o_3$  und  $o_4$  verfügbar sind. Tabelle 4.1 zeigt welche Informationen über die Operationen der anderen Geräte an ein Gerät, unter Verwendung verschiedener Ansätze, jeweils übertragen wird. Die diagonalen Einträge sind in den Tabellen jeweils ausgegraut, da es für dieses Beispiel nicht relevant ist, welche Informationen die Geräte über sich selbst erhalten. Tabelle 4.1a zeigt dabei die Informationen, welche die Geräte besitzen, wenn der naive Ansatz verwendet wird, bei dem jedes Gerät alle Informationen über alle anderen Geräte erhält. Beispielsweise erhält  $d_1$  die Information, dass auf  $d_2$  die Operationen  $o_2$  und  $o_5$  und auf Gerät  $d_3$  die Operationen  $o_3$  und  $o_4$  verfügbar sind. Bei diesem Ansatz würde jedes Gerät auch das komplette Datenflussmodell aus Abbildung 4.4 erhalten.



**Abbildung 4.4:** Graph des Datenflussmodells für das verwendete Beispiel

Gerät \ kennt	$d_1$	$d_2$	$d_3$
$d_1$		$o_2, o_5$	$o_3, o_4$
$d_2$	$o_1, o_6$		$o_3, o_4$
$d_3$	$o_1, o_6$	$o_2, o_5$	

(a) naiver Ansatz

Gerät \ kennt	$d_1$	$d_2$	$d_3$
$d_1$		$o_2, o_5$	X
$d_2$	$o_1, o_6$		$o_3, o_4$
$d_3$	X	$o_2, o_5$	

(b) Geräte kennen nur Nachfolger

Gerät \ kennt	$d_1$	$d_2$	$d_3$
$d_1$		$o_2$	X
$d_2$	$o_6$		$o_3$
$d_3$	X	$o_5$	

(c) Geräte kennen nur relevante Operationen der Nachfolger

**Tabelle 4.1:** Tabellarische Darstellung der Informationen, die die Geräte über andere Geräte bzw. deren Operationen unter Anwendungen der beschriebenen Optimierungen besitzen

Die Definition der Datenflussmodelle aus 2.1 wird um die Abbildung

$$\delta: O \rightarrow \hat{O} \tag{4.13}$$

erweitert, wobei  $\hat{O}$  die Menge der Namen der Operationen ist.  $\delta$  bildet dabei eine Operation  $o \in O$  aus dem Graphen auf den Namen dieser Operation  $\hat{o} \in \hat{O}$  ab. Dazu sei

$$\begin{aligned} \delta': \hat{O} &\rightarrow \mathcal{P}(O) \\ a &\mapsto \{b \in O \mid \delta(b) = a\} \end{aligned} \tag{4.14}$$

die Abbildung, die den Namen einer Operation auf eine Menge von Operationen abbildet, die diesen Namen tragen. Die Differenzierung zwischen  $O$  und  $\hat{O}$  ist notwendig, da jede Operation im Datenflussmodell einmalig ist, aber dennoch mehrere dieser Operationen den selben Namen besitzen können. Ich definiere eine weitere Nachfolgerfunktion  $sucOp$ , die analog zu  $suc$  (Gleichung (2.4)) funktioniert, allerdings auf den Namen der Operationen statt auf den Operationen selbst arbeitet:

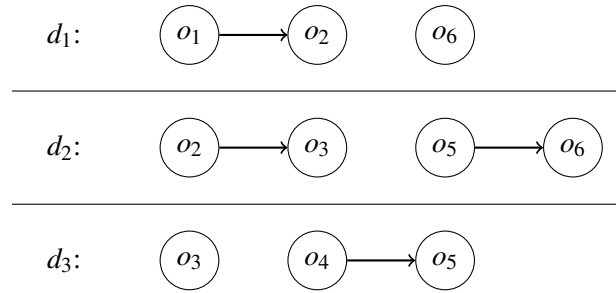
$$\begin{aligned} sucOp: \hat{O} &\rightarrow \mathcal{P}(\hat{O}) \\ a &\mapsto \{\delta(b) \in \hat{O} \mid o \in \delta'(a) \wedge b \in suc(o)\} \end{aligned} \tag{4.15}$$

Des Weiteren sei  $D$  die Menge der Geräte auf denen die ME ausgeführt wird und

$$\gamma: D \rightarrow \mathcal{P}(\hat{O}) \tag{4.16}$$

eine Abbildung, die von einem Gerät  $d \in D$  auf eine Menge von Namen der Operationen  $\hat{O}_d \subseteq \hat{O}$  abbildet, wobei dies die Namen der Operationen sind, die auf diesem Gerät ausgeführt werden können. Dazu sei

$$\begin{aligned} \gamma': \hat{O} &\rightarrow \mathcal{P}(D) \\ a &\mapsto \{b \in D \mid \gamma(b) = a\} \end{aligned} \tag{4.17}$$



**Abbildung 4.5:** Die Teilgraphen  $DF_d$  des Datenflussmodells, die die Geräte aus dem Beispiel jeweils erhalten

eine Abbildung, die auf Eingabe des Namen einer Operation, die Geräte ermittelt, die Operationen mit diesem Namen ausführen können.

$$\begin{aligned} \text{sucD}: D &\rightarrow \mathcal{P}(D) \\ a &\mapsto \{\delta(b) \in D \mid c \in \gamma(a) \wedge d \in \text{sucOp}(c) \wedge b \in \gamma'(d)\} \end{aligned} \quad (4.18)$$

$$\begin{aligned} \text{preD}: D &\rightarrow \mathcal{P}(D) \\ a &\mapsto \{b \in D \mid a \in \text{sucD}(b)\} \end{aligned} \quad (4.19)$$

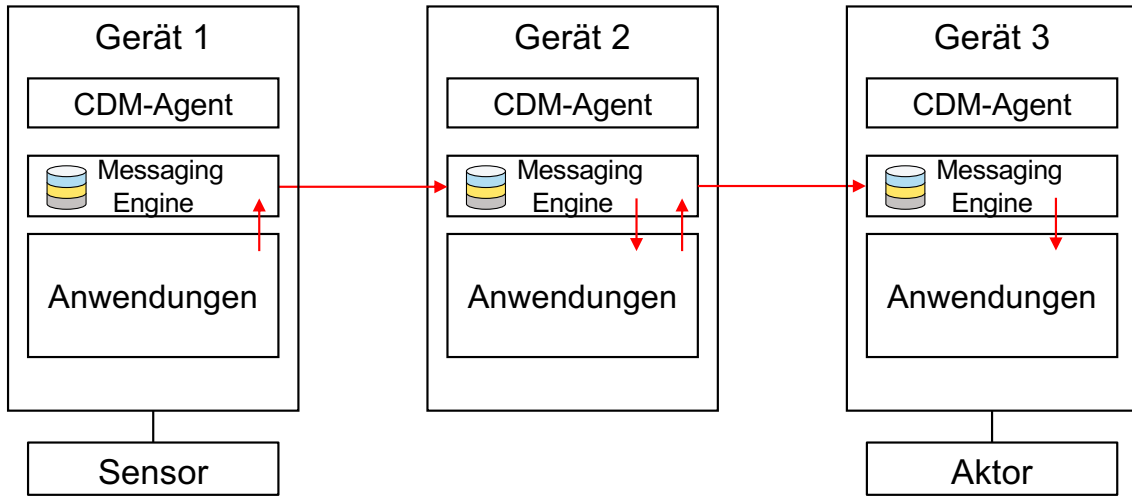
sind Funktionen, um für ein Gerät  $d \in D$  die korrespondierenden Geräte bestimmen zu können, die Operationen ausführen, die Vor- bzw. Nachfolger von den Operationen sind, die von  $d$  ausgeführt werden.

Ist  $d_{\text{neu}} \in D$  nun das Gerät, das neu in der Umgebung ist, so müssen nur die Geräte aus  $\text{preD}(d_{\text{neu}})$  Informationen über  $d_{\text{neu}}$  erhalten und  $d_{\text{neu}}$  benötigt lediglich Informationen über die Geräte aus  $\text{sucD}(d_{\text{neu}})$ . Tabelle 4.1b zeigt entsprechend, welche Informationen die Geräte aus dem Beispiel bei diesem Ansatz entsprechend über die anderen Geräte besitzen würden. Die beiden Zellen mit einem X bedeuten, dass dieses Gerät keinerlei Informationen über das andere Gerät besitzen muss. Im Vergleich zum naiven Ansatz (Tabelle 4.1a) erhält Gerät  $d_3$  nun keinerlei Informationen über Gerät  $d_1$ , ebenso wie  $d_1$  keinerlei Informationen über  $d_3$  erhält.

Doch ein Gerät benötigt grundsätzlich nicht die vollständige Liste der Namen der anwendbaren Operationen eines anderen Geräts [DH19]. Es werden nur Informationen über die Namen der Operationen des anderen Geräts benötigt, das tatsächlich Operationen ausführt, die Nachfolgeoperationen der Operationen des ersten Geräts sind. Die Menge der Namen der Operationen, die ein Gerät  $t \in D$  ausführen kann, und für ein anderes Gerät  $s \in D$  relevant ist, lässt sich formalisieren als

$$\begin{aligned} \text{relevantOp}: D \times D &\rightarrow \mathcal{P}(\hat{O}) \\ (s, t) &\mapsto \{b \in \text{sucOp}(c) \cap \gamma(t) \mid c \in \gamma(s)\} \end{aligned} \quad (4.20)$$

Mittels der Funktionen  $\text{preD}$ ,  $\text{sucD}$  und  $\text{relevantOp}$  lässt sich also bestimmen, welche Geräte voneinander Kenntnis benötigen und welche Namen von Operationen für jene Geräte bekannt sein müssen. Tabelle 4.1c zeigt wieder die Informationen, die die Geräte nun über andere Geräte erhalten. Im Vergleich zu Tabelle 4.1b kennen die Geräte nun nur noch die für sie relevanten Operationen der anderen Geräte. Bereits in diesem Beispiel mit nur drei Geräten und sechs Operation ist im Vergleich zu Tabelle 4.1a deutlich die Reduzierung der auf den Geräten gespeicherten Informationen sichtbar. Damit die ME Daten verarbeiten kann, benötigen die Geräte aber auch noch die Informationen über die in  $DF$  (Def. 2.1) definierten Datenflussmodelle. Nach dem naiven Ansatz erhalten alle



**Abbildung 4.6:** Veranschaulichung der Datenverarbeitung mithilfe der Messaging Engine

drei Geräte aus dem Beispiel das vollständige Datenflussmodell aus Abbildung 4.4. Auch dabei ist es im allgemeinen aber nicht notwendig, dass jedes Gerät den gesamten Graph kennt [DH19]. Ein Gerät benötigt nur den Teilgraph, der die Übergänge von den Operationen des Geräts auf die nachfolgenden Operationen enthält. Dieser Teilgraph kann für ein Gerät  $d \in D$  wie folgt beschrieben werden:

$$DF_{d \in D} := (O_d \subseteq O, E_d \subseteq E) \quad (4.21)$$

mit den Operationen

$$O_d := O_d^s \cup O_d^t \quad (4.22)$$

$$O_d^s := \{o \in \delta'(c) | c \in \gamma(d)\} \quad (4.23)$$

$$O_d^t := \{o \in \text{suc}(c) | c \in O_d^s\} \quad (4.24)$$

wobei  $O_d^s$  die Operationen von  $d$ ,  $O_d^t$  die darauf folgenden Operationen und

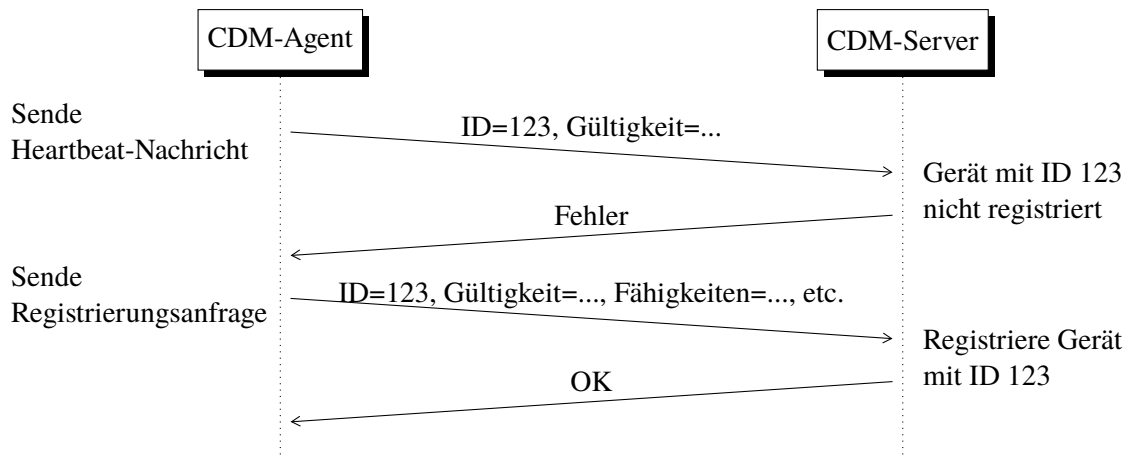
$$E_d := \{(a, b) \in E | a \in O_d^s \wedge b \in O_d^t\} \quad (4.25)$$

die Kanten zwischen diesen sind. Dabei wird angenommen, dass der Graph  $DF$  alle Datenflussmodelle der Umgebung enthält. Dies ist möglich da  $DF$  nicht zusammenhängend sein muss. Damit beschreibt  $DF_d$  den Teilgraph, der für ein Gerät  $d \in D$  tatsächlich relevant ist. Wird auf jedes Gerät jeweils nur dieser relevante Teilgraph übertragen, reduziert sich die Datenmenge, die über das Netzwerk übertragen und auf den Geräten gespeichert werden muss. In Abbildung 4.5 ist für die drei Geräte aus dem Beispiel jeweils der resultierende Teilgraph zu sehen. Die entstandenen Teilgraphen sind im Vergleich zum kompletten Graph aus Abbildung 4.4 für die Geräte  $d_1$  und  $d_3$  um mehr als die Hälfte reduziert.

## 4.7 Schritt 6: Datenverarbeitung und Monitoring

Da die ME auf allen Geräten für die Umgebung konfiguriert wurde, können Daten nach den Definitionen der Datenflussmodelle dezentral auf den Geräten verarbeitet werden. Wenn ein Gerät aus der Umgebung entfernt wird, werden unter Umständen zur Verarbeitung ausstehende Daten





**Abbildung 4.7:** Grafische Darstellung des Ablaufs: Der CDM-Server erhält eine Heartbeat-Nachricht von einem unbekanntem Gerät

vom Gerät in eine Datenbank des CDM-Servers übertragen (siehe Abschnitt 4.8.1). Deshalb prüft der CDM-Server nun, ob auf dem Gerät Operationen verfügbar sind, für die der Server unverarbeitete Daten in der Datenbank zwischengespeichert hat. Liegen solche Daten vor, werden sie aus der Datenbank entfernt und an das Gerät zur Verarbeitung übertragen. Abbildung 4.6 dient der Veranschaulichung der Datenverarbeitung mithilfe der ME. Gerät 1 liest Daten von einem angeschlossenen Sensor aus. Gerät 2 verfügt über eine Operation, um die Daten des Sensors auszuwerten, sodass damit ein Aktor gesteuert werden kann. Deshalb sendet Gerät 1 diese Daten an Gerät 2. Die ME auf Gerät 2 übergibt diese Daten an die installierte Anwendung zur Auswertung der Daten. Die Anwendung verarbeitet die Daten und gibt der ME das Ergebnis zurück. Da an Gerät 2 kein Aktor angeschlossen ist, kann dieses Gerät mit den produzierten Daten nichts anfangen. An Gerät 3 ist aber ein Aktor angeschlossen. Deshalb leitet Gerät 2 diese Daten an Gerät 3 weiter. Die ME auf Gerät 3 gibt diese Daten dann an die Anwendung weiter, die für die Steuerung des Aktors zuständig ist.

#### 4.7.1 Heartbeat

Damit der CDM-Server erkennen kann, wenn ein Gerät die Umgebung unerwartet verlässt, sendet der CDM-Agent periodisch Nachrichten an den Server. Diese Nachrichten werden als Heartbeat (dt. Herzschlag) bezeichnet, weil der Server weiß, dass ein Gerät noch da ist und noch lebt, solange er diese Nachrichten empfängt. Wenn das Gerät bereits registriert ist, kennt der CDM-Server alle nötigen Informationen über das Gerät. Diese Heartbeat-Nachrichten sind deshalb eine abgespeckte Variante der Registrierungsanfragen, welche lediglich die ID des Geräts und die Gültigkeit der Anfrage enthalten. Erhält der CDM-Server eine Heartbeat-Nachricht von einem nicht registrierten Gerät, lehnt er diese ab. Dadurch weiß der CDM-Agent, dass das Gerät nicht registriert ist und eine vollständige Registrierungsanfrage mit allen notwendigen Informationen über das Gerät an den Server gesendet werden muss. Dieser Ablauf ist in Abbildung 4.7 dargestellt.

Empfängt der CDM-Server von einem Gerät keine Heartbeat-Nachrichten mehr, womit die Registrierung ungültig wird, kann er automatisch eine Warnmeldung an eine konfigurierte E-Mail Adresse versenden, um den Administrator darüber zu informieren. Ebenfalls könnte er das Simple

Network Management Protocol (SNMP) [Pre02] verwenden, um eine Warnmeldung an eine zentrale Instanz zu übertragen. SNMP ist ein weitverbreitetes Netzwerkprotokoll zur zentralen Überwachung und Steuerung von Netzwerkkomponenten.

### 4.7.2 Monitoring

Der CDM-Agent kann auch dazu verwendet werden um die Geräte zu überwachen. So kann er Daten über die Geräte wie CPU-Auslastung, Speicherbelegung, Netzwerkauslastung, usw. sammeln, filtern und aggregieren. Erst wenn die gemessenen Werte einen Grenzwert übersteigen oder sich ungewöhnlich schnell verändern, werden diese an den CDM-Server gesendet. Dadurch wird ein Teil der Rechenleistung, die zum Interpretieren der Messwerte benötigt wird, vom CDM-Server auf die Geräte ausgelagert und zudem auch Netzwerkverkehr eingespart. Dies führt zu mehr Skalierbarkeit in der IoT-Umgebung [ABDP13]. Diese Werte können dann über Schnittstellen am CDM-Server abgerufen werden. Aber auch hier könnten andere Kommunikationsmethoden wie E-Mail oder SNMP verwendet werden, um den Administrator zu informieren.

## 4.8 Schritt 7: Geräte entfernen

Im letzten Schritt wird das Verlassen eines Geräts aus der Umgebung behandelt. Dabei gibt es zwei Möglichkeiten: (i) Das Gerät verschwindet unerwartet aus der Umgebung, zum Beispiel wegen eines Defekts. Dabei gehen unverarbeitete Daten verloren, die sich zu diesem Zeitpunkt auf dem Gerät befinden. Das Verschwinden des Geräts wird dabei durch das Ausbleiben des Heartbeat erkannt. (ii) Zum anderen ist es möglich das Gerät geplant aus der Umgebung zu entfernen. Dazu bekommt der CDM-Server im Voraus die Anweisung, das Gerät aus der Konfiguration zu entfernen, während es noch erreichbar ist. Dadurch bietet sich die Möglichkeit, die Datenverarbeitung auf dem Gerät einzustellen und unverarbeitete Daten zu sichern.

### 4.8.1 Erwartetes Entfernen

Um das Gerät ordnungsgemäß aus der Umgebung zu entfernen, wird der CDM-Server über die Registrierungsschnittstelle darüber informiert, dass das Gerät entfernt werden soll. Das Entfernen eines Geräts aus der Umgebung kann entweder von dem Gerät selbst angestoßen werden oder manuell über eine Nachricht an die Registrierungsschnittstelle des CDM-Servers. Das Gerät kann das Entfernen zum Beispiel auslösen, wenn es heruntergefahren werden soll oder wenn ein entsprechendes Skript auf dem Gerät ausgeführt wird, das über die MBP gestartet werden kann. So ist es möglich, das Entfernen eines Geräts über die Nutzeroberfläche der MBP anzustoßen. Wird das Entfernen des Geräts angestoßen, vermerkt der CDM-Server in der Gerätedatenbank, dass das Entfernen des Geräts eingeleitet wurde. Damit wird sichergestellt, dass andere Geräte, die zeitgleich konfiguriert werden, keine Informationen über dieses Gerät erhalten. Dann entfernt der CDM-Server die Informationen über dieses Gerät von den anderen Geräten der Umgebung. Anschließend fordert der CDM-Server die ME auf dem Gerät dazu auf, die Verarbeitung unverarbeiteter ausstehender Daten einzustellen. Dann werden diese ausstehenden Daten durch dem CDM-Server nacheinander abgerufen. Der CDM-Server ermittelt für jedes Datum die verfügbaren Geräte, die zur Weiterverarbeitung in der Lage sind. Sind solche Geräte verfügbar, werden die Daten vom Server

direkt an die jeweiligen Geräte weitergeleitet. War dieses Gerät allerdings das einzige Gerät der Umgebung, das bestimmte Operationen ausführen konnte, gibt es keine anderen Geräte, an die die Daten zur Verarbeitung übertragen werden können. Dies sollte nur in Ausnahmefällen auftreten, da dies bedeutet, dass nicht alle Datenflussmodelle korrekt ausgeführt werden können. Der fehlerfreie Betrieb der Umgebung ist somit nicht sichergestellt. Dennoch werden diese Daten in diesem Fall in einer Datenbank, den jeweiligen Operationen zugeordnet, auf dem Server gespeichert. Dort werden sie so lange zwischengespeichert, bis der Umgebung ein Gerät beitrifft, das in der Lage ist, diese Daten zu verarbeiten. Die Daten werden dann von dem CDM-Server auf dieses Gerät zur Verarbeitung übertragen. Nachdem alle Daten von dem zu entfernenden Gerät extrahiert wurden, wird das Gerät aus der MBP entfernt. Dabei entfernt die MBP die installierte Software von dem Gerät. Anschließend wird das Gerät aus der Gerätedatenbank der Umgebung entfernt.

### 4.8.2 Unerwartetes Entfernen

Werden von dem Gerät keine Heartbeat-Nachrichten mehr empfangen, sodass der Gültigkeitszeitraum der zuletzt empfangenen Nachrichten überschritten wird, geht der CDM-Server davon aus, dass das Gerät nicht mehr im Netzwerk der Umgebung präsent ist. Da die ME die Erreichbarkeit eines Geräts prüft bevor Daten zur Weiterverarbeitung an dieses weitergeleitet werden, stellt es für die Verarbeitung neuer Daten kein Problem dar, solange es noch andere Geräte gibt, die die Operationen ebenfalls ausführen können. Da das Gerät nicht mehr erreichbar ist, können allerdings die auf dem Gerät zur Verarbeitung ausstehenden Daten nicht mehr gesichert werden. Zudem sind die Daten je nach Kontext nicht mehr von Bedeutung, falls das Gerät nach einiger Zeit wieder erreichbar werden sollte. Um das Gerät zu entfernen, wird es von dem CDM-Server aus der MBP entfernt und die Informationen über das Gerät werden von den anderen Geräten der Umgebung entfernt.

## 4.9 Mögliche Optimierungen

Das beschriebene Konzept bietet einige Möglichkeiten zur Optimierung. So geht das Konzept beispielsweise nicht darauf ein, wie der CDM-Server damit umgeht, wenn Softwareartefakte in der MBP aktualisiert werden. Außerdem geht das Konzept nicht näher darauf ein, wie die Installation und Deinstallation von Software auf den Geräten gehandhabt wird, damit die installierte Software sich nicht gegenseitig beeinflusst oder sogar stört. Diese Themen sollten behandelt werden, damit ein stabiler Betrieb der Umgebung über längere Zeit gewährleistet werden kann. Daher werden in diesem Kapitel mögliche Optimierungen vorgestellt, die diese beiden Themen behandeln.

### 4.9.1 Softwarekontrolle

Bei der Registrierung eines neuen Geräts muss die benötigte Software über das Netzwerk auf das Gerät übertragen und dort installiert werden. Je nach Software und Gerät benötigt die Installation auf dem Gerät einige Zeit. Außerdem ist es zeitaufwendig die Datenmengen über das Netzwerk zu übertragen. War das Gerät schon einmal Teil der Umgebung und verfügt bereits über die benötigte Software, dann wird diese bei der Registrierung erneut installiert und die vorherige Installation entfernt. Dieses Verhalten ließe sich durch ein Konzept zur Verwaltung der von der Umgebung installierten Software vermeiden. Die Geräte könnten dazu in der Registrierungsanfrage

Informationen zu der bereits installierten Software an den CDM-Server übermitteln, sodass nur die Software erneut übertragen und installiert werden muss, die noch nicht auf dem Gerät installiert ist. Gibt es beispielsweise ein Problem mit der Stromversorgung und viele der Geräte der Umgebung wären einige Zeit ohne Stromversorgung, dann könnte in dieser Zeit kein Heartbeat gesendet werden. Somit werden diese Geräte von dem CDM-Server aus der Konfiguration entfernt. Wenn die Geräte nun wieder gestartet werden, dann muss die gesamte benötigte Software auf jedes einzelne Gerät erneut übertragen und installiert werden. Dadurch kann es unter Umständen lange dauern bis die Umgebung wieder vollständig einsatzbereit ist. Außerdem wird das Netzwerk unnötig belastet. Diese Optimierung hätte allerdings nicht nur in solchen Ausnahmesituationen einen Mehrwert, sondern auch in Szenarien, in denen Geräte situationsbedingt nicht ununterbrochen mit der Umgebung verbunden sind, zum Beispiel weil diese nachts abgeschaltet oder ortsveränderlich sind.

Ein weiteres Problem besteht in der fehlenden Versionierung der Software. Damit der Betrieb der Umgebung funktioniert, muss die installierte Software auf allen Geräten auf dem gleichen Stand sein. Um dies sicherzustellen muss der Server dazu in der Lage sein die unterschiedlichen Versionen voneinander zu unterscheiden. Wird das Softwareartefakt in der MBP aktualisiert, wird die neue Version auf zukünftig neu registrierte Geräte installiert; auf bereits vorhandenen Geräten wird jedoch weiterhin die vorherige Version verwendet. Würde man das Konzept der Softwarekontrolle noch um Informationen zur installierten Version erweitern, könnte der CDM-Server jeweils die auf den Geräten installierte Version mit der geforderten Version abgleichen und diese falls nötig aktualisieren.

Konkret könnten Hashwerte verwendet werden, um unterschiedliche Versionen der Softwareartefakte zu unterscheiden. Dazu könnte in der MBP jedes mal ein Hashwert des Softwareartefakts berechnet und mit diesem verknüpft in einer Datenbank gespeichert werden. Wenn das Softwareartefakt auf ein Gerät installiert wird, könnte dieser Hashwert ebenfalls auf dem Gerät hinterlegt werden. Der CDM-Agent könnte dann in den Heartbeat-Nachrichten einen kumulierten Hashwert dieser Hashwerte an den CDM-Server übertragen. Der Server könnte diesen Hashwert mit dem von ihm erwarteten Hashwert vergleichen. Sind diese unterschiedlich besitzt mindestens ein Softwareartefakt nicht die erwartete Version. Der Server könnte nun über den CDM-Agent eine Liste mit der installierten Software und den jeweiligen Hashwerten anfordern um dadurch genau das Softwareartefakt zu ermitteln, welches aktualisiert werden muss. Wird in den Heartbeat-Nachrichten eine vollständige Liste der installierten Softwareartefakte übertragen, steigt die Größe der Nachrichten mit zunehmender Anzahl der installierten Softwareartefakte. Wird in den Heartbeat-Nachrichten hingegen der kumulierte Hashwert verwendet, hängt die Größe der Nachrichten nicht von der Anzahl der installierten Softwareartefakte ab.

### 4.9.2 Containervirtualisierung

Mit steigender Anzahl an Operationen, und damit steigender Anzahl an Software, die auf ein Gerät installiert wird, steigt die Gefahr, dass sich die Programme gegenseitig beeinträchtigen und dabei ungewollte Seiteneffekte auftreten. Dies kann zum Beispiel passieren, wenn zwei Applikationen unterschiedliche Versionen derselben Bibliothek benötigen, die nicht miteinander kompatibel sind. Zudem müssen die Operationen selbst dafür sorgen, dass bei ihrer Deinstallation auch jegliche Software entfernt wird, die von ihr benötigt wurde. Damit eine Applikation die von ihr benötigte andere Software entfernen kann, muss sie feststellen können, ob diese Software weiterhin von anderen auf dem Gerät installierten Operationen benötigt werden. Da sich dies unter Umständen nicht ohne weiteres sicher feststellen lässt, verbleibt dann entweder nicht benötigte Software

auf dem Gerät oder Software wird entfernt, die noch von anderen Operationen benötigt wird, sodass diese nur noch teilweise oder überhaupt nicht mehr funktionieren. Um diese Problematik zu vermeiden könnte Containervirtualisierung verwendet werden. Die Containervirtualisierung verwendet sogenannte Container, um Applikationen voneinander abzukapseln. Auf einem Gerät, dem sogenannten Host, können gleichzeitig mehrere Container ausgeführt werden. Ein Container verhält sich aus Sicht einer darin installierten Applikation wie ein reguläres Betriebssystem. Die Applikation kann dort wie gewohnt ihre benötigten Abhängigkeiten installieren. Diese werden allerdings nicht direkt auf dem Host-Betriebssystem installiert, sondern auf einem virtuellen Dateisystem des Containers. Dadurch beeinflusst sich die installierte Software in den Containern nicht gegenseitig. So könnte für jede Operation ein eigener Container auf dem Gerät verwendet werden, in dem die jeweilige Software installiert wird. Da der Container alle nötigen Informationen zum Ausführungskontext der enthaltenen Applikationen besitzt, ist es auch möglich, Container mit der bereits fertig installierten Software auf die Geräte zu übertragen. Dies hat den Vorteil, dass die meist ressourcenintensive Installation nicht auf den IoT-Geräten ausgeführt werden muss, sodass es weniger zeitintensiv ist, neue Geräte in die Umgebung einzubeziehen. Wird eine Operation auf einem Gerät nicht mehr benötigt, so kann einfach der jeweilige Container vom Gerät entfernt werden. Dadurch wird vermieden, dass nicht weiter benötigte Abhängigkeiten auf dem Gerät verbleiben oder weiter benötigte Abhängigkeiten fälschlicherweise entfernt werden. Durch die Verwendung des Containers als zusätzliche Ebene zwischen Host-Betriebssystem und der Applikationen ist allerdings mit verringerter Performance der Applikationen zu rechnen. Morabito hat in seinem Paper „Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation“ den Einfluss auf die Performance unter Verwendung von Docker<sup>1</sup>-Containern in unterschiedlichen Szenarien auf typischen IoT-Geräten, wie dem Raspberry Pi 2 und 3, analysiert [Mor17]. Konkret können Docker-Container verwendet werden, die von der MBP auf die Geräte übertragen und mithilfe der Adapter-Skripte gestartet und beendet werden können.

---

<sup>1</sup>[www.docker.com](http://www.docker.com)



## 5 Implementierung

In diesem Kapitel wird eine Implementierung des in Kapitel 4 beschriebenen Konzeptes vorgestellt. Diese prototypische Implementierung dient dazu, die praktische Umsetzbarkeit des Konzeptes zu evaluieren. Der CDM-Server ist in der Programmiersprache Java in Version 8 implementiert, da Java auf unterschiedlichen Betriebssystemen verwendet werden kann und dafür das Californium-Framework<sup>1</sup> verfügbar ist, mit welchem einfach CoAP-Schnittstellen implementiert werden können. Californium ist ein leistungsstarkes CoAP-Framework, das für den Einsatz auf Servern konzipiert ist [KLS14]. Die gewählte Programmiersprache für den CDM-Agenten auf den Geräten ist Python in Version 3.6, da Python ebenfalls für viele Betriebssysteme verfügbar ist und die Implementierung so auf unterschiedlichen Geräten eingesetzt werden kann. Ebenso ist auch die ME in Python implementiert, weswegen auf den Geräten bereits eine Python-Umgebung benötigt wird. Für die CoAP Kommunikation wird serverseitig das Californium-Framework und auf Agentenseite die Bibliothek CoAPthon [TVM15] verwendet. Zur serverseitigen Datenspeicherung wird die dokumentenorientierte Datenbank MongoDB<sup>2</sup> verwendet. Bei der prototypischen Entwicklung bietet diese den Vorteil gegenüber relationalen Datenbanken, dass die Struktur der zu speichernden Daten nicht strikt festgelegt werden muss. Zur Übertragung von Daten über CoAP werden die Datenformate JavaScript Object Notation (JSON) [Bra17] oder Concise Binary Object Representation (CBOR) [BH13] verwendet. JSON ist ein in Webanwendungen häufig genutztes und weit verbreitetes Datenformat. JSON ist von Menschen lesbar, da Textzeichen zur Darstellung der Daten verwendet werden. CBOR ist ein an JSON angelehntes Datenformat, das die Daten binär codiert, wodurch es für Menschen nicht lesbar ist. Zur Serialisierung und Deserialisierung dieser Daten werden Bibliotheken des Jackson Project<sup>3</sup> verwendet. Die MBP sieht vor, dass zur automatisierten Installation der Applikationen auf den Geräten Bash-Skripte zum Einsatz kommen.

Zum Testen der Implementierung wurden Raspberry Pis und virtuelle Maschinen (VMs) verwendet. Raspberry Pis sind kleine Einplatinencomputer, die für den Betrieb lediglich noch ein USB-Netzteil benötigen. Der Raspberry Pi in Modell 3B besitzt einen Vierkern-ARM-Prozessor und einem Gigabyte Arbeitsspeicher<sup>4</sup>. Als Betriebssystem wurde Raspbian<sup>5</sup> in Version 8 verwendet, ein Debian-basiertes Linux-Betriebssystem, das für die Verwendung auf dem Raspberry Pi optimiert ist. Die VMs waren jeweils mit zwei Prozessorkernen und zwei Gigabyte Arbeitsspeicher ausgestattet. Als Betriebssystem wurde die Linux-Distribution Ubuntu<sup>6</sup> in Version 18.04 LTS verwendet. Zum Testen wurden auf einer der VMs eine Instanz des CDM-Servers und eine Instanz der MBP installiert und gestartet. Auf den anderen VMs und den Raspberry Pis wurde jeweils der CDM-Agent installiert. Als der CDM-Agent auf den Geräten gestartet wurde, wurde der CDM-Server gefunden, die Geräte

---

<sup>1</sup><https://www.eclipse.org/californium/>

<sup>2</sup><https://www.mongodb.com>

<sup>3</sup><https://github.com/FasterXML/jackson>

<sup>4</sup><https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>

<sup>5</sup><https://www.raspbian.org>

<sup>6</sup><https://ubuntu.com>

haben sich registriert und wurden korrekt eingerichtet. Um die korrekte Konfiguration der ME zu prüfen wurde ein Datenflussmodell mit vier Operationen verwendet, die an eingehende Texte den Namen des Geräts anhängen und diesen dann an die ME zurückgeben. Der entstandene Text wurde ebenfalls in einer Textdatei gespeichert. Mit dieser Textdatei konnte dann geprüft werden, ob die Nachrichten über den erwarteten Pfad über die Geräte geleitet wurden.

## 5.1 CDM-Agent

Der CDM-Agent muss auf allen Geräten ausgeführt werden, die der Umgebung hinzugefügt werden sollen. Damit der CDM-Agent nach dem Start eines Geräts ebenfalls gestartet wird, ist er auf den Geräten als Dienst installiert. Dabei handelt es sich um einen systemd-Dienst [JG15]. Der CDM-Agenten auf den Geräten verfügt über eine Konfigurationsdatei mit einer Vorkonfiguration. Darin sind beispielsweise die Fähigkeiten des Geräts festgelegt.

### 5.1.1 CDM-Server-Discovery

Die CDM-Server-Discovery besteht aus zwei Teilen: Zuerst wird die CoAP-Multicast-Service-Discovery verwendet, um die Adresse des CDM-Servers und den Pfad zur Registrierungsschnittstelle zu ermitteln. Falls das Netzwerk keine Multicast-Kommunikation unterstützt oder erlaubt, funktioniert diese Methode nicht. In diesem Fall wird eine andere Methode benötigt. Dazu wurden bereits zwei alternative Methoden in den Abschnitten 4.2.3 und 4.2.3 beschrieben. Da die Umsetzung der DHCP-Methode nicht praktikabel ist, wird in der Implementierung die beschriebene DNS-Methode verwendet. Dabei wird die Adresse des CDM-Servers mithilfe eines DNS-Servers aufgelöst.

**CoAP-Multicast-Service-Discovery** Ein CoAP-Server kann die von ihm angebotenen Ressourcen für andere CoAP-Endpunkte auffindbar machen. Dazu stellt er eine Beschreibung seiner angebotenen Ressourcen unter dem Pfad „/.well-known/core“ zur Verfügung. Für die Beschreibung der Ressourcen wird eine Auflistung im CoRE Link Format verwendet. Dabei können die Einträge Attribute besitzen mit denen die Eigenschaften der Ressourcen beschrieben werden. Eine GET-Anfrage an den Pfad „/.well-known/core“ an den CDM-Server liefert die folgende Liste:

```

</.well-known/core >,
</flows >; ct=50,
< /registry >; ct=50 ; rt="registry-resource" ; title="device registry resource"

```

Pfad zur Ressource
Genutztes Format der Schnittstelle
Typ-Attribut
Menschenlesbarer Titel

Die Einträge der Liste sind durch Kommas und die Attribute mit Semikolons voneinander getrennt. In den spitzen Klammern ist jeweils der Pfad der Ressource angegeben. Das ct-Attribut (content type) gibt das Format an, das von der Schnittstelle verwendet wird. Der Wert 50 ist JSON zugeordnet. Das rt-Attribut (resource type) enthält eine eindeutige Zeichenkette, die den Typ der Ressource beschreibt. Der CDM-Server verwendet „registry-resource“ für das Typ-Attribut der Registrierungsschnittstelle. Der CDM-Agent durchsucht diese Liste nach einem Eintrag vom Typ „registry-resource“ und erhält so den Pfad zur Registrierungsschnittstelle. Allerdings ist es in CoAP



auch möglich, bereits in der Anfrage nach bestimmten Eigenschaften zu filtern. Um nach dem Typ der Registrierungsschnittstelle zu filtern, hängt der CDM-Agent den Suffix „?rt=registry-resource“ an den Pfad an. Auf eine GET-Anfrage an „/.well-known/core?rt=registry-resource“ an den CDM-Server erhält er dann eine Liste, die nur den Eintrag der Registrierungsschnittstelle enthält.

Bis jetzt kennt der CDM-Agent allerdings die Netzwerkadresse des CDM-Servers noch nicht, daher wird Multicast verwendet. In CoAP sind dazu „All CoAP Nodes“-Multicast-Adressen [RFC7252, S. 97] festgelegt, auf die alle CoAP-Endpunkte horchen können, die ihre Ressourcen mithilfe der CoAP-Multicast-Service-Discovery auffindbar machen wollen. Der CDM-Agent sendet deshalb die GET-Anfrage an „/.well-known/core?rt=registry-resource“ an diese Multicast-Gruppen. Da der CDM-Server Teil dieser Multicast-Gruppe ist, erhält der die Anfrage und antwortet darauf wie bereits beschrieben. Andere CoAP-Server im Netzwerk, die diese Anfrage erhalten, besitzen keine Ressource vom Typ „registry-resource“, sodass die gefilterte Liste leer ist. Diese Filterfunktion wird allerdings von der Spezifikation als optional angegeben, sodass sie eventuell nicht von allen CoAP-Server im Netzwerk unterstützt wird [RFC6690, S. 11]. Im Idealfall senden CoAP-Server gar keine Antwort, wenn die Liste leer ist. Aber auch dieses Verhalten wird von der Spezifikation als optional angesehen [RFC7252, S. 66]. Der CDM-Agent durchsucht die erhaltenen Antworten nach Einträgen vom Typ „registry-resource“ und verwendet die Adresse dieses Servers zusammen mit der Angabe zum Pfad der Ressource aus dem Eintrag. Erhält der CDM-Agent keine Antworten oder findet in den erhaltenen Antworten keine Ressource vom Typ „registry-resource“, so wird davon ausgegangen, dass in dem Netzwerk keine Multicast-Kommunikation verfügbar ist. Dies ist in einigen großen Netzwerken, wie auch im Netzwerk der Universität Stuttgart, der Fall. Unter diesen Umständen wird die Server-Discovery mittels DNS verwendet.

**DNS-Discovery** Um auch in Netzwerken ohne Multicast-Unterstützung eine Verbindung zum Server aufbauen zu können, wird alternativ DNS verwendet, um die Adresse des CDM-Servers zu ermitteln. Dazu muss dem Gerät über die Netzwerkkonfiguration ein DNS-Server bekannt sein, der die Adresse des CDM-Servers der Umgebung kennt und aus der Umgebung erreichbar ist. Hierfür gibt es zwei Möglichkeiten: Entweder wird pro Umgebung ein separater DNS-Server verwendet oder es wird ein spezieller DNS-Server verwendet, der für mehrere Umgebungen zuständig ist. Dieser erkennt aus welcher Umgebung eine Anfrage gesendet wurde und antwortet darauf mit der Adresse des zu dieser Umgebung zugehörigen CDM-Servers. Dieser DNS-Server kennt die Adresse des CDM-Servers in der Umgebung, sodass diese über eine feste Domain aufgelöst werden kann. Der Pfad, an dem der CDM-Agent die Registrierungsschnittstelle des CDM-Servers erwartet, ist fest definiert.

### 5.1.2 Gerät registrieren

Nachdem der CDM-Agent die URI der Registrierungsschnittstelle des CDM-Servers in Erfahrung gebracht hat, wird eine Registrierungsanfrage an diese Schnittstelle gesendet. Listing 5.1 zeigt ein Beispiel für eine solche Registrierungsanfrage in JSON-Repräsentation. Der angegebene Name muss in der Umgebung eindeutig sein, da er zur Identifizierung des Geräts verwendet wird. Um einen eindeutigen Namen zu erzeugen, wird die MAC-Adresse des Geräts an den in der Konfigurationsdatei des Geräts festgelegten Präfix angehängt. Um die zu installierenden Anwendungen für ein Gerät zu bestimmen wird der Einfachheit halber eine vereinfachte Variante des in Abschnitt 4.4 beschriebenen Verfahrens verwendet. Dabei sind die kompatiblen Operatoren für ein Gerät bereits auf diesem

### Listing 5.1 Beispielhafte Registrierungsanfrage in JSON-Repräsentation

---

```
1 {
2   "device":{
3     "name":"temp_acbc3279f91b",
4     "operations":[
5       "TemperatureSensor",
6       "TemperatureFilter"
7     ],
8     "address":"192.168.56.11",
9     "mac":"ac:bc:32:79:f9:1b",
10    "credentialGroup":"group1"
11  },
12  "options":{
13    "ttl":3600
14  }
15 }
```

---

vorkonfiguriert. Dazu beinhaltet die Konfigurationsdatei eine Liste mit Strings, wobei jeder String für eine Operation steht. Diese Liste ist Teil der Anfrage. Unabhängig von den Operatoren wird die ME auf jedes Gerät installiert. Die Netzwerkadresse des Geräts kann in der Anfrage explizit enthalten sein, ist aber optional, da der CDM-Server alternativ die Adresse aus den Absenderinformationen des Pakets verwendet. Zum Zugriff auf das Gerät benötigt der CDM-Server die passenden Anmeldeinformationen. Da sich die Anmeldeinformationen zwischen den Geräten unterscheiden können, besitzt der CDM-Server eine Liste mit den verschiedenen Anmeldeinformationen, wobei diesen jeweils ein Name zugeordnet ist. Das `credentialGroup`-Attribut der Anfrage enthält einen dieser Namen, sodass der CDM-Server weiß, welche Anmeldeinformationen zum Zugriff auf das Gerät nötig sind. Die MAC-Adresse wird beim Anlegen des Geräts in der MBP benötigt und deshalb ebenfalls übermittelt. Der `options`-Teil der Anfrage enthält die `Time to live` in Sekunden. Diese gibt an, wie lange die Anfrage ihre Gültigkeit behalten soll. Damit die Registrierung des Geräts nicht ungültig wird, sendet der CDM-Agent eine erneute Anfrage nach dem 90 % dieser Zeit abgelaufen sind. Der Einfachheit halber wird dabei jedes mal eine vollständige Anfrage gesendet, anstatt die in Abschnitt 4.7.1 beschriebene verkürzte Version zu verwenden, was allerdings keine Einschränkung darstellt.

#### 5.1.3 Start der Applikationen

Nachdem eine Anwendung von der MBP auf dem Gerät installiert wurde, wird diese automatisch gestartet. Damit die installierten Anwendungen auch nach einem Neustart des Geräts ebenso wieder gestartet werden, müssten sich diese bei der Installation, zum Beispiel als Dienst, auf dem Gerät installieren. Um die Installation einfacher zu gestalten, prüft der CDM-Agent beim Start, ob sich im Installationsverzeichnis von einer der Anwendungen ein Skript mit dem Namen `"autostart.sh"` befindet und startet dieses, wenn es existiert. Das Skript sorgt dafür, dass die Anwendung gestartet wird. Jede Anwendung, die auf diese Weise automatisch gestartet werden soll, muss ein `"autostart.sh"`-Skript in ihrem Installationsordner besitzen. Durch diesen Mechanismus wird beispielsweise die ME beim Start des CDM-Agent ebenfalls gestartet.

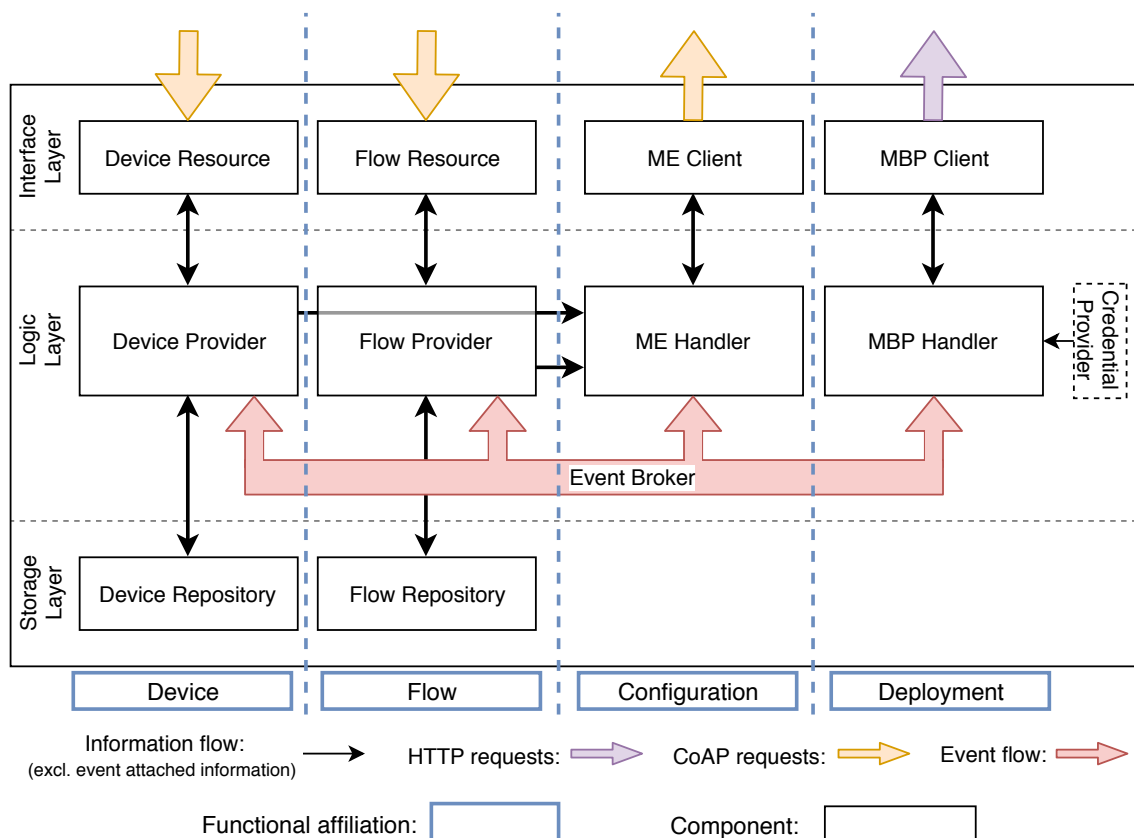


Abbildung 5.1: Übersicht der Architektur des CDM-Servers

## 5.2 CDM-Server

In diesem Abschnitt wird auf den Aufbau des CDM-Servers in der Implementierung eingegangen. Der Server kommuniziert sowohl mit der ME als auch mit dem CDM-Agent auf den Geräten ebenso wie mit der MBP. Die Komponenten des CDM-Servers aus Abbildung 5.1 können horizontal in drei Schichten unterteilt werden: Netzwerkschnittstellen (*Interface Layer*), Anwendungslogik (*Logic Layer*) und Speicher (*Storage Layer*). Über die eingehenden Schnittstellen, *Device Resource* und *Flow Resource*, werden die auf dem Server gespeicherten Datenflussmodelle und Geräteinformationen verwaltet. Die ausgehenden Schnittstellen, *MBP Client* und *ME Client*, werden genutzt, um mit der MBP und der Messaging Engine zu kommunizieren. Die Anwendungslogik interpretiert die ankommenden Informationen über Geräte und Datenflussmodelle und verarbeitet diese. Die Speicherkomponenten werden von der Anwendungslogik benutzt, um die Informationen über Geräte und Datenflussmodelle persistent zu speichern.

Ebenso können die Komponenten aus Abbildung 5.1 auch vertikal anhand ihrer Funktionen unterteilt werden: Verwaltung der Geräte (*Device*), Verwaltung der Datenflussmodelle (*Flow*), Konfiguration der Geräte (*Configuration*) und Software-Deployment (*Deployment*). Die Logikkomponenten des Servers arbeiten nebenläufig, da Aufgaben, wie das Installieren der Applikationen über die MBP einige Minuten dauern können, sodass die Nebenläufigkeit verhindert, dass der gesamte CDM-Server während dieser Zeit blockiert wird. Außerdem können mehrere Registrierungsanfragen gleichzeitig vom Server empfangen werden, weswegen es sinnvoll ist, auch Anfragen der CDM-Agenten

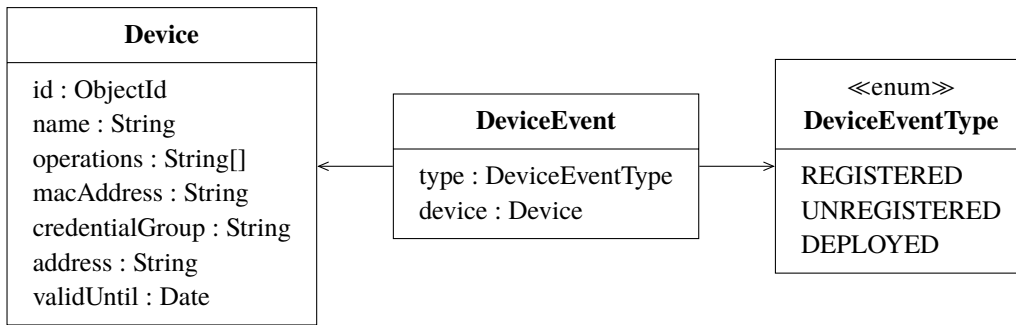


Abbildung 5.2: UML-Diagramm von einem Geräte-Event

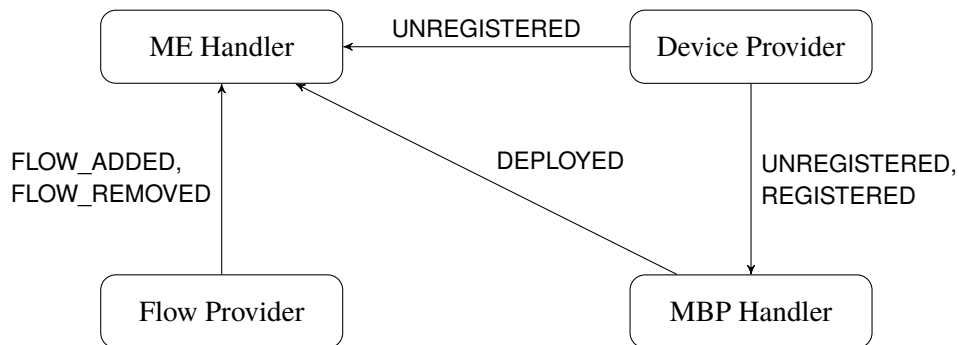


Abbildung 5.3: Grafische Darstellung des Event-Austauschs zwischen den Komponenten des CDM-Servers

nebenläufig zu bearbeiten. Um Informationen zwischen den Komponenten zu übertragen, ohne die aufrufende Komponente dabei zu blockieren, werden Events verwendet, die über den *Event Broker* an die jeweiligen Komponenten weitergeleitet werden. Die *Event Broker*-Komponente nimmt Events entgegen und leitet diese an Komponenten weiter, die sich zuvor für diesen Typ von Events registriert haben. Die Implementierung kennt zwei Typen von Events: Geräte-Events und Datenflussmodell-Events. Ersteres bezieht sich immer auf ein Gerät, letzteres auf ein Datenflussmodell. Beide Typen besitzen dazu noch eine Information über die Art des Event. Bei auf Geräte bezogene Events kann dieses »Software auf dem Gerät installiert« (*DEPLOYED*), »Gerät registriert« (*REGISTERED*) oder »Gerät entfernt« (*UNREGISTERED*) sein. Für Datenflussmodell bezogene Events gibt es nur die beiden Arten »Datenflussmodell hinzugefügt« (*FLOW\_ADDED*) und »Datenflussmodell entfernt« (*FLOW\_REMOVED*).

Abbildung 5.2 zeigt das UML-Diagramm eines Gerät-Events. Das Event gehört zu genau einem Gerät und besitzt genau einen Typ. Das *id*-Attribut der *Device*-Klasse wird von der Datenbank erzeugt und nur von dieser verwendet.

Abbildung 5.3 zeigt zwischen welchen Komponenten welche Events ausgetauscht werden. Darin sind die Kästen die Komponenten und die Pfeile stellen die Pfade der Events dar. Komponenten mit ausgehenden Pfeilen erstellen die Events mit denen die Pfeile beschriftet sind. Die Komponenten mit eingehenden Pfeilen empfangen diese Events. Die Events werden aber nicht direkt zwischen den Komponenten übertragen, sondern vom *Event Broker* übermittelt.

### 5.2.1 Geräteverwaltung

Die Geräteverwaltung setzt sich aus drei Komponenten zusammen: Der CoAP-Schnittstelle, um Geräte zu registrieren oder zu entfernen, der Anwendungslogik und der Komponente zur Speicherung der Informationen in der Datenbank. Diese drei Komponenten sind in der Spalte *Device* in Abbildung 5.1 zu sehen.

**Device Resource** Der CDM-Server stellt in der *Device Resource*-Komponente eine Schnittstelle zur Verfügung, um die registrierten Geräte zu verwalten. Über diese Schnittstelle können mit PUT neue Geräte angelegt, bzw. die Gültigkeit der Registrierung für registrierte Geräte verlängert werden. Mit DELETE können Geräte entfernt und mit GET Informationen über registrierte Geräte abgefragt werden. Listing 5.1 zeigt den Nachrichteninhalte einer Registrierungsanfrage an die Geräteschnittstelle in der JSON-Darstellung. Die Schnittstelle verwendet CoAP als Übertragungsprotokoll. Der Nachrichteninhalte kann mit JSON oder CBOR codiert sein. Der Inhalt der Nachricht wird zu einem Java-Objekt deserialisiert und dieses Objekt zur weiteren Verarbeitung an die Anwendungslogik übergeben. Bei PUT repräsentiert dieses Objekt die Registrierungsanfrage, bei DELETE den Namen des Geräts als String. Bei GET-Anfragen kann optional nach Geräten gefiltert werden, die einen bestimmten Namen haben oder die Operationen mit bestimmten Namen ausführen können.

**Device Provider** Die *Device Provider*-Komponente besitzt die Anwendungslogik für die Geräteverwaltung. Erhält der *Device Provider* das Objekt mit den Informationen der Registrierungsanfrage, so wird geprüft, ob bereits ein Gerät mit diesem Namen angelegt ist. Ist das Gerät bereits registriert, wird die Anfrage als Heartbeat gewertet und lediglich die Gültigkeit der Registrierung, entsprechend der Anfrage, in der Datenbank erneuert. Ist das Gerät noch nicht registriert, so wird die Registrierung in der Datenbank gespeichert und ein »Gerät registriert«-Event, mit den Informationen zu dem Gerät, an die *Event Broker*-Komponente übergeben. Dieses Event wird dann vom *MBP Handler* (siehe Abschnitt 5.2.3) weiterverarbeitet.

Ein *Timer Task*<sup>7</sup> ist eine in Java verfügbare Klasse, um Aufgaben periodisch auszuführen. In der Implementierung wird ein *Timer Task* verwendet, um die Registrierungen in der Datenbank regelmäßig auf ihre zeitliche Gültigkeit zu prüfen. Werden dabei abgelaufene Registrierungen gefunden, werden diese aus der Datenbank entfernt und für jede abgelaufene Registrierung ein »Gerät entfernt«-Event mit den Informationen zum Gerät ausgelöst.

**Device Repository** Die *Device Repository*-Komponente wird dazu verwendet, die gespeicherten Informationen über die Geräte und deren Registrierungen zu verwalten. Dazu besitzt die Komponente Methoden um Geräte abzurufen, anzulegen, zu löschen sowie die gespeicherten Informationen zu aktualisieren. Außerdem besitzt die Komponente eine Methode, um die Gültigkeit der Registrierungen der gespeicherten Geräte zu überprüfen. Diese Methode löscht Geräte mit ungültiger Registrierung aus der Datenbank und gibt die Liste der gelöschten Geräte zurück.

<sup>7</sup><https://docs.oracle.com/javase/8/docs/api/java/util/TimerTask.html>

### Listing 5.2 Beispielhaftes Datenflussmodell in JSON-Repräsentation

```

1  {
2    "flow": {
3      "A": {
4        "operation": "op1",
5        "next_oiid_list":["B","C"]
6      },
7      "B": {
8        "operation": "op2",
9        "next_oiid": "D"
10     },
11     "C": {
12       "operation": "op3",
13       "next_oiid": "D"
14     },
15     "D": {
16       "operation": "op4",
17       "join_list":["B","C"],
18       "next_oiid": "none"
19     }
20   },
21   "flow_id": "sample"
22 }

```

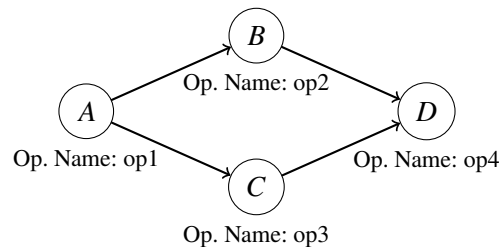


Abbildung 5.4: Datenflussmodell aus Listing 5.2 als Graph dargestellt

## 5.2.2 Datenflussmodellverwaltung

Die Datenflussmodellverwaltung ist weitgehend analog zur Geräteverwaltung aufgebaut. Auch sie verfügt über die drei Komponenten: Die CoAP-Schnittstelle, um Datenflussmodelle anzulegen, bzw. zu entfernen (*Flow Resource*), die Anwendungslogik (*Flow Provider*), die die Modelle auf Konsistenz prüft und anderen Komponenten den Zugriff auf die Modelle ermöglicht und die *Flow Repository*-Komponente, um den Zugriff auf die Datenbank zu verwalten in der die Modelle gespeichert werden. Diese Komponenten sind in der zweiten Spalte von links (*Functional Affiliation Flow*) in Abbildung 5.1 zu sehen.

**Flow Resource** Analog zu der Geräteverwaltungsschnittstelle besitzt der CDM-Server mit der *Flow Resource*-Komponente eine CoAP-Schnittstelle, um die Datenflussmodelle der Umgebung zu verwalten. Dabei ist es möglich, Datenflussmodelle mit PUT anzulegen, mit GET registrierte Modelle abzurufen und Modelle mit DELETE zu entfernen. Das Format zur Darstellung der Datenflussmodelle wurde aus der Implementierung der ME übernommen. So lässt sich die Datenflussschnittstelle des CDM-Servers analog zu der Datenflussschnittstelle der ME verwenden. Listing 5.2 zeigt ein Datenflussmodell in der JSON-Darstellung, wie sie vom CDM-Server und der ME verwendet wird. Abbildung 5.4 zeigt dasselbe Datenflussmodell als Graph dargestellt. Der Datenfluss teilt sich in dem Modell an Knoten A auf zwei parallele Datenflüsse auf. Diese werden von B und C verarbeitet und laufen an Knoten D wieder zusammen. Das Datenflussmodell in der JSON-Darstellung besitzt zwei Attribute: *flow\_id* und *flow*. *flow\_id* gibt den Namen des Datenflussmodells an. Dieser muss eindeutig sein und wird dazu verwendet um das Datenflussmodell beispielsweise beim Entfernen

zu referenzieren. Das `flow`-Attribut enthält eine Datenstruktur, welche die einzelnen Operationen beschreibt. Jede Operation besitzt eine ID. Die Datenstruktur ordnet jeder ID ein Objekt zu, das diese Operation im Kontext des Datenflussmodells beschreibt. Jedes dieser Operationsobjekte besitzt ein Attribut `operation`, das den Namen der Operation angibt. Damit wird definiert, welche Applikation verwendet wird, um diese Operation auszuführen. Da sich der Datenfluss an Knoten *A* aufteilt, besitzt das Operationsobjekt von *A* ein Attribut `next_oid_list` mit einer Liste der folgenden Operationen. Knoten *B* und *C* besitzen jeweils nur eine Nachfolgeroperation. Deshalb besitzen diese ein Attribut `next_oid` mit der Nachfolgeroperation. Der Knoten *D* besitzt keine Nachfolgeroperation. Daher ist das Attribut `next_oid` für diese Operation mit dem Wert `none` belegt. Außerdem laufen an Operation *D* die beiden parallelen Datenflüsse wieder zusammen. Aus diesem Grund besitzt es ein Attribut `join_list` mit einer Liste der beiden Vorgängeroperationen.

**Flow Provider** Erhält die *Flow Provider*-Komponente ein Datenflussmodell um dieses anzulegen, so wird es erst auf Konsistenz geprüft. Ein Modell kann nur angelegt werden, wenn es konsistent und die ID nicht bereits vergeben ist. Trifft dies nicht zu, so wird die Anfrage abgelehnt. Andernfalls wird das Modell in der Datenbank gespeichert und ein »Datenflussmodell hinzugefügt«-Event mit Informationen zu dem neuen Modell an die *Event Broker*-Komponente übergeben. Analog dazu wird ein »Datenflussmodell entfernt«-Event mit den Informationen zum entfernten Modell generiert. Diese Events werden vom *ME Handler* verarbeitet, wie in Abschnitt 5.2.4 beschrieben. Die Komponente dient auch dazu, die Datenflussmodelle zu interpretieren, um die Vor- und Nachfolgeroperationen für eine Operation zu bestimmen.

**Flow Repository** Die *Flow Repository*-Komponente regelt den Zugriff auf die Datenbank, um die Datenflussmodelle zu verwalten. Sie besitzt Methoden, um Datenflussmodelle abzurufen, zu speichern und zu löschen. Die Datenflussmodelle werden in der Datenbank in der selben Form wie das Datenflussmodell aus Listing 5.2 gespeichert.

### 5.2.3 Software-Deployment

Das Software-Deployment und das Bestimmen der Applikationen wird vom *MBP Handler* und dem *MBP Client* durchgeführt. Diese Komponenten sind in Abbildung 5.1 in der rechten Spalte (*Functional Affiliation Deployment*) zu sehen. Mithilfe von Threads kann der Programmfluss eines Programms aufgeteilt werden, sodass diese Teile unabhängig voneinander, gleichzeitig ausgeführt werden können. Um viele kurzlebige Threads zu verwalten, können Threadpools eingesetzt werden. Kurzlebige Threads sind in der Regel solche, die erstellt werden, um eine Aufgabe abzuarbeiten und sich dann beenden, wenn die Aufgabe abgeschlossen ist. In Java sorgt ein *ThreadPoolExecutor*<sup>8</sup> dafür, dass ein Threadpool mit begrenzter Größe immer voll ausgelastet ist, solange weitere Threads in einer Warteschlange auf ihre Ausführung warten. Der *MBP Handler* verwendet so einen *ThreadPoolExecutor*, um eine begrenzte Anzahl an Aufgaben parallel zu bearbeiten. Erhält er ein »Gerät registriert« oder ein »Gerät entfernt«-Event, so wird ein neuer Thread für das Bereitstellen oder das Entfernen des Geräts erstellt und dem *ThreadPoolExecutor* übergeben. Hat der Threadpool bereits seine maximale Anzahl an auszuführenden Threads erreicht, werden weitere Threads in eine

<sup>8</sup><https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html>

FIFO-Warteschlange abgelegt. FIFO steht für First-In-First-Out und bedeutet ,dass die Elemente aus der Warteschlange in der Reihenfolge herausgenommen werden, in der sie hinzugefügt wurden. Der *MBP Handler* besitzt die Liste `REQUIRED_ADAPTER_NAMES` mit Namen von Adaptern, die unabhängig von den vom Gerät mitgeteilten Operationen auf jedes Gerät installiert werden sollen. Diese Liste enthält den Eintrag für den Adapter der ME. Durch die Verwendung der Liste bietet sich die Möglichkeit auf diese Weise auch andere Adapter, zum Beispiel mit Funktionalität zum erweiterten Monitoring der Geräte, zu installieren. Die *MBP Handler*-Komponente enthält den Threadpool und die Logik, die *MBP Client*-Komponente dient dazu, mit der HTTP-Schnittstelle der MBP zu kommunizieren und die übertragenen Daten zu serialisieren und zu deserialisieren. Zum Bereitstellen wird das Gerät zuerst bei der MBP registriert.

Die für den Zugriff auf das Gerät nötigen Anmeldeinformationen werden über das `credentialGroup`-Attribut des Geräts mithilfe der *Credential Provider*-Komponente bestimmt. Da die genaue Art der Speicherung der Anmeldeinformationen keine Auswirkung auf die Umsetzbarkeit des Konzepts hat, ist diese Komponente nur rudimentär implementiert, weswegen sie in Abbildung 5.1 auch nur gestrichelt angedeutet ist. Die Anmeldeinformationen bestehen aus einem Nutzernamen, einem Passwort und einem kryptografischen Schlüssel. Diese Informationen werden mit anderen Informationen über das Gerät, wie dessen Namen, MAC-Adresse und IP-Adresse, an die MBP zur Registrierung des Geräts übertragen. Der CDM-Server erhält die ID, die die MBP zur Referenzierung des Geräts verwendet. Ebenso verwendet die MBP IDs zur Referenzierung der Adapter, bietet aber eine Schnittstelle an, um Adapter über ihren Namen zu finden. Da der CDM-Server zunächst nur die Namen der benötigten Adapter, nicht aber die ID kennt, müssen die Namen zu den IDs aufgelöst werden. Um die Anzahl der Anfragen an die MBP zu reduzieren, verwendet der CDM-Server einen Cache, in dem die Zuordnung der Namen der Adapter zu ihren IDs zwischengespeichert wird. Nur wenn noch kein Eintrag für einen Adapter im Cache vorliegt, muss die MBP zur Auflösung des Namens kontaktiert werden. Da die MBP nur Sensoren oder Aktoren als Typen für installierbare Komponenten kennt, werden alle zu installierenden Adapter als Sensor auf dem Gerät an der MBP angelegt, was allerdings keine Einschränkung des Konzepts darstellt. Zum Anlegen des Sensors wird eine Anfrage mit der ID des zu installierenden Adapters und der ID des Geräts an die MBP gesendet und der CDM-Server erhält dabei die ID des neu angelegten Sensors. Schließlich wird eine weitere Anfrage mit dieser ID an die MBP gesendet, um die Software auf das Gerät zu übertragen, zu installieren und auf dem Gerät zu starten. Dazu wird auf dem Gerät ein Verzeichnis angelegt, dessen Name die ID des Sensors beinhaltet. Dann die Dateien für den Adapter in dieses Verzeichnis übertragen, anschließend zuerst das `"install.sh"`-Skript zur Installation und danach das `"start.sh"`-Skript zum Starten der Applikation ausgeführt. Auf diese Weise werden zuerst die in `REQUIRED_ADAPTER_NAMES` genannten Adapter, also die ME, dann für jede einzelne vom Gerät mitgeteilte Operation ein gleichnamiger Adapter installiert und gestartet. Ist die Installation aller Adapter auf einem Gerät abgeschlossen, wird ein »Software auf dem Gerät installiert«-Event für das installierte Gerät an die *Event Broker*-Komponente übergeben, wodurch die Konfiguration der ME angestoßen wird.

Die ME muss für jede Operation jeweils den Pfad kennen, an dem die Operation installiert wurde. Da der Pfad aber die ID des Sensors beinhaltet, unterscheidet sich dieser für die selbe Operation von Gerät zu Gerät. Um der ME auf jedem Gerät trotzdem für jede Operation einen festen Einstiegspunkt zu bieten, werden symbolische Verknüpfungen verwendet. Eine symbolische Verknüpfung ist eine Datei im Dateisystem, die auf eine andere Datei oder ein anderes Verzeichnis verweist. Über die Verknüpfung kann so auf das verknüpfte Verzeichnis zugegriffen werden, als wäre es dort gespeichert. Dazu wird zum Ende der Installation ein Skript mit dem Namen `"symlink.sh"` aufgerufen, das unter dem Pfad `"/opt/OP"` eine symbolische Verknüpfung zum Installationsverzeichnis anlegt, die den



Namen der Operation trägt. Wird beispielsweise die Operation `TemperatureFilter` auf einem Gerät installiert, dann findet sich nach der Installation unter `/opt/OP/TemperatureFilter` ein Verweis auf das tatsächliche Installationsverzeichnis.

Soll ein Gerät entfernt werden, wird eine Anfrage an die MBP verschickt, um die Registrierung des Geräts in der MBP zu löschen. Die MBP versucht daraufhin auf dem Gerät für jeden installierten Adapter das `stop.sh`-Skript aufzurufen. Diese Skripte haben die Aufgabe, die jeweilige Anwendung ordnungsgemäß zu beenden und dafür zu sorgen, dass nicht verarbeitete Daten, wie zum Beispiel Nachrichten in der ME, extrahiert und auf anderen Geräte weiterverarbeitet werden können. Anschließend löscht die MBP das Verzeichnis der Anwendung. Hat das Gerät die Umgebung allerdings bereits verlassen, wurde also ohne Ankündigung entfernt, so kann die MBP das Gerät zu diesem Zeitpunkt nicht mehr erreichen. Die Software verbleibt auf dem Gerät und die unverarbeiteten Daten können nicht extrahiert werden. Wird das Gerät zu einem späteren Zeitpunkt wieder an dem CDM-Server registriert, werden neue Sensoren an der MBP angelegt, die andere IDs haben und so würden weiteren Instanzen der Software in Verzeichnissen neben den alten Instanzen der vorherigen Installation angelegt werden. Dabei würden die symbolischen Verknüpfungen zu den alten Installationen mit Verknüpfungen zu den neuen Installationen ersetzt werden. Damit die Anzahl dieser Überreste durch vorherige Installationen nicht weiter zunimmt und dadurch der Speicher des Geräts vollständig aufgebraucht wird, wird vor dem Anlegen einer symbolischen Verknüpfung für eine Operation geprüft, ob bereits eine solche Verknüpfung existiert. Ist dies der Fall, so wird das Verzeichnis mit der alten Installation, auf das die Verknüpfung zeigt, entfernt und dann erst die Verknüpfung zu der neuen Installation angelegt.

### 5.2.4 Konfiguration Messaging Engine

Die Konfiguration der ME auf den Geräten wird vom *ME Handler* und vom *ME Client* vorgenommen. Diese sind in der Spalte rechts der Mitte (*Configuration*) in Abbildung 5.1 zu sehen. Allerdings wird der Kern der Logik zum Finden der Vor- und Nachfolgegeräte, wie bereits in Abschnitt 5.2.2 erläutert, vom *Flow Provider* zur Verfügung gestellt. Der *ME Handler* reagiert auf die folgenden Events, die ebenfalls in Abbildung 5.3 dargestellt sind: Wurde die Software auf einem neuen Gerät installiert, erhält der *ME Handler* ein »Software auf dem Gerät installiert«-Event. Wird ein Gerät entfernt, erhält er ein »Gerät entfernt«-Event. Wenn die Definition eines Datenflussmodells über die Datenflussschnittstelle hinzugefügt oder entfernt wurde, erhält der *ME Handler* entsprechend ein »Datenflussmodell hinzugefügt« bzw. »Datenflussmodell entfernt«-Event.

Wie der *MBP Handler* verwendet auch der *ME Handler* einen *ThreadPoolExecutor*, um die Anzahl der gleichzeitig ablaufenden Aufgaben zu begrenzen. Der CoAP-Client des verwendeten Californium-Frameworks unterstützt sowohl synchrone als auch asynchrone Funktionsaufrufe für das Senden von CoAP-Nachrichten. Beide Techniken werden vom *ME Client* verwendet. Bei synchronen Funktionsaufrufen muss das Programm so lange auf die aufgerufene Funktion warten, bis diese abgearbeitet wurde und ein Ergebnis zurückliefert. Beim Senden einer Nachricht gibt dieses Ergebnis Auskunft darüber, ob die Nachricht erfolgreich zugestellt werden konnte. Damit der CoAP-Client des Senders weiß, ob eine Nachricht erfolgreich zugestellt wurde, sendet der CoAP-Client des Empfängers bei Erhalt einer Nachricht eine Empfangsbestätigung. Wird zum Senden einer Nachricht ein synchroner Funktionsaufruf verwendet, muss das Programm an dieser Stelle im Programmcode also so lange warten, bis der CoAP-Client die Empfangsbestätigung erhalten hat, oder das Senden der Nachricht aufgibt, da er nach wiederholten Sendeversuchen keine Empfangsbestätigung erhalten hat. Bei Verwendung der Standardparameter dauert es laut

CoAP-Spezifikation 93 Sekunden bis der CoAP-Client den Sendevorgang aufgibt [RFC7252, S. 29]. Werden hingegen asynchrone Funktionsaufrufe zum Senden einer Nachricht verwendet, ist das Programm nicht dazu gezwungen an dieser Stelle im Programmcode auf das Ergebnis des Sendevorgangs zu warten. Die Information über die erfolgreiche Zustellung ruft das Programm zu einem späteren Zeitpunkt ab. In der Zwischenzeit kann es weitere Nachrichten versenden oder andere Aufgaben erledigen, die nicht von der Zustellung vorheriger Nachrichten abhängen. Muss eine Information auf viele Geräte übertragen werden, wie zum Beispiel eine Information über neue Geräte oder Datenflussmodelle, bietet es sich an, dafür die asynchronen Funktionsaufrufe zu verwenden. Dabei werden viele Nachrichten verschickt und dann erst auf die Empfangsbestätigung aller Nachrichten gewartet. Die Wartezeit entspricht dann nämlich nur der Zeit, bis die letzte Bestätigung eingegangen ist. Werden hingegen synchrone Funktionsaufrufe verwendet, um die vielen Nachrichten zu versenden, wird die nächste Nachricht erst gesendet, wenn das Ergebnis des vorherigen Sendevorgangs bekannt ist. Dabei entspricht die gesamte Wartezeit dann der Summe der Wartezeiten der einzelnen Sendevorgänge.

Sollen allerdings viele Informationen auf ein einziges Gerät übertragen werden, kann es durchaus von Nachteil sein, eine große Anzahl an Nachrichten gleichzeitig zu verschicken. Kann das empfangende Gerät nicht alle Nachrichten gleichzeitig verarbeiten, verbleiben die überschüssigen Nachrichten in einer Warteschlange. Werden mehr Nachrichten empfangen, als in der Warteschlange gespeichert werden können, wird das Gerät überlastet und weitere Nachrichten gehen verloren. Für diese verlorenen Nachrichten wird keine Bestätigung empfangen, sodass darauf vergebens gewartet wird, bevor ein erneuter Sendeversuch unternommen wird. Da in CoAP eine Überlastkontrolle spezifiziert ist, sollte die verwendete CoAP-Implementierung selbst dafür sorgen, dass die Anzahl der gleichzeitig übertragenen Nachrichten je Empfänger einen festgelegten Wert nicht übersteigt. In CoAP liegt dieser Wert standardmäßig bei 1. Zum Zeitpunkt der Implementierung hatte die Überlastkontrolle allerdings in dem verwendeten Californium-Framework nicht fehlerfrei funktioniert und musste explizit aktiviert werden. Müssen viele Nachrichten an denselben Empfänger übertragen werden, werden die Nachrichten aus diesem Grund sequenziell gesendet.

Erhält der *ME Handler* ein »Software auf dem Gerät installiert«-Event, so werden mithilfe des *Flow Providers* die Nachfolgegeräte des Geräts, die Vorgängergeräte des Geräts und die für dieses Gerät relevanten Datenflussmodelle bestimmt. Um die Informationen über das Gerät auf die Vorgängergeräte zu übermitteln, werden asynchrone Methoden des *ME Client* verwendet. Dadurch muss nicht direkt auf die erfolgreiche Übertragung gewartet werden, stattdessen werden die Informationen über die relevanten Datenflussmodelle, die Informationen über die Nachfolgegeräte und die Informationen über die auf dem Gerät installierten Operationen der Reihe nach auf das Gerät übertragen. Erst dann wird, falls notwendig, darauf gewartet, dass die Informationen an die Vorgängergeräte erfolgreich übertragen werden.

Erhält der *ME Handler* ein »Gerät entfernt«-Event, dann werden die asynchronen Methoden des *ME Client* verwendet, um die Informationen über das entfernte Gerät von den anderen Geräten der Umgebung zu entfernen. Dazu sendet der *ME Client* entsprechende CoAP-Anfragen die ME auf den Geräten.

Wird ein Datenflussmodell hinzugefügt oder entfernt erhält der *ME Handler* ein »Datenflussmodell hinzugefügt« oder »Datenflussmodell entfernt«-Event. Bei einem »Datenflussmodell hinzugefügt«-Event bestimmt der *ME Handler* zuerst die Geräte, die die Operationen ausführen können, die in dem neuen Datenflussmodell enthalten sind. Auf diese Geräte wird dann mithilfe der asynchronen Methoden des *ME Client* das neue Datenflussmodell übertragen. Anschließend werden für jede Operation die Vorgängeroperationen in dem neuen Datenflussmodell bestimmt. Dann werden die Geräte bestimmt, die die Operation ausführen können. Informationen über diese Geräte werden

dann über den *ME Client* auf die Geräte übertragen, die die Vorgängeroperationen ausführen können. Dadurch erhält jedes Gerät die notwendigen Informationen über die Nachfolgergeräte für die Ausführung des neuen Datenflussmodells. Bei einem »Datenflussmodell entfernt«-Event werden ebenfalls die Geräte bestimmt, die die Operationen des zu entfernenden Datenflussmodells ausführen können. Anschließend wird dieses Datenflussmodell mithilfe des *ME Client* von diesen Geräten entfernt.



## 6 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde ein Konzept entwickelt mit dem IoT-Geräte automatisiert in eine Umgebung integriert und verwaltet werden können. Die Geräte sollen so eingerichtet werden, dass die Umgebung die Ressourcen der Geräte nutzen kann. Der Vorgang der Integration neuer Geräte wurde in einzelne Schritte unterteilt und diese jeweils erläutert. Um die Umsetzbarkeit zu prüfen, wurde ein Prototyp implementiert.

Immer mehr elektronische Geräte können mit dem Internet verbunden werden und sind dadurch Teil des Internet of Things (dt. Internet der Dinge, kurz IoT). Damit diese Geräte miteinander arbeiten können, müssen sie füreinander konfiguriert sein. So müssen neue Geräte zuerst für eine Umgebung eingerichtet werden, damit sie in dieser verwendet werden können. Ebenso müssen die anderen Geräte der Umgebung auf das neue Gerät eingerichtet werden. Allerdings gibt es auch ortsveränderliche Geräte, wie beispielsweise Smartphones und Fahrzeuge, die häufig kommen und gehen, sodass jedes mal eine Neueinrichtung erforderlich ist. Aufgrund dieser häufigen Neueinrichtung ist eine manuelle Konfiguration nicht praktikabel. Zudem wären viele Nutzer damit überfordert, die Geräte manuell zu konfigurieren. Außerdem sollen die Geräte eine Vereinfachung des täglichen Lebens bewirken. Deshalb wird ein System benötigt, das den Nutzern diese Konfiguration abnimmt. Dieses System muss neue Geräte integrieren können und die bestehenden Geräte verwalten. Außerdem muss es darauf reagieren, wenn ein Gerät die Umgebung verlassen hat. Im Konzept dieser Arbeit wird deshalb ein Server vorgestellt, der diese Verwaltung und Integration übernimmt. Für die Datenverarbeitung und Datenübertragung wird eine Messaging Engine verwendet. Da die Messaging Engine mit Datenflussmodellen arbeitet, besitzt der Server eine Datenbank, um diese zu speichern. Um ein neues Gerät zu integrieren muss dieses den Server finden und sich an diesem Server registrieren. Der Server installiert dann passende Software auf dem Gerät, damit das Gerät für die Umgebung nutzbar wird. Diese Prozedur zur Integration habe ich in meinem Konzept in sieben Schritte unterteilt: 1. Schritt: Das neue Gerät findet den Server. 2. Schritt: Das neue Gerät registriert sich am Server und die Informationen über das Gerät werden dort in einer Datenbank gespeichert. 3. Schritt: Der Server wählt die passende Software für das Gerät, anhand dessen Fähigkeiten, aus. 4. Schritt: Die ausgewählte Software wird auf das Gerät installiert. 5. Schritt: Die Messaging Engine auf den Geräten wird konfiguriert. 6. Schritt: Die Geräte arbeiten zusammen. Außerdem senden sie regelmäßig Heartbeat-Nachrichten an den Server, damit dieser weiß, dass die Geräte noch da sind. Schritt 7 tritt ein, wenn ein Gerät nicht mehr da ist oder entfernt werden soll. Das Gerät wird aus der Konfiguration der anderen Geräte entfernt. Werden Datenflussmodelle am Server entfernt oder hinzugefügt, so werden diese Modelle vom Server ebenfalls auf den Geräten entfernt oder hinzugefügt.

Der Prototyp des Servers (genannt CDM-Server) wurde in der Programmiersprache Java implementiert, da Java plattformunabhängig verwendet werden kann. Auf den Geräten wird der sogenannte CDM-Agent eingesetzt. Dieser ist in der Programmiersprache Python implementiert, da Python ebenfalls für viele Betriebssysteme verfügbar ist und die Messaging Engine ebenfalls darin implementiert ist. Zur Kommunikation zwischen CDM-Agent und CDM-Server wird das

Protokoll CoAP eingesetzt, da es für den Einsatz auf ressourcenschwachen Geräten optimiert ist. Der CDM-Agent verwendet die CoAP-Multicast-Service-Discovery oder alternativ eine Methode über DNS, um den CDM-Server in der Umgebung zu finden. Um die Software auf die Geräte zu installieren, wird die Multi-purpose Binding and Provisioning Platform (MBP) eingesetzt.

### **Ausblick**

Da das Konzept von einer einzigen Instanz des CDM-Servers pro Umgebung ausgeht, ist die Skalierbarkeit begrenzt und zudem bildet der Server einen Single-Point-of-Failure. Das Konzept sollte daher um Mechanismen erweitert werden, mit denen in einer Umgebung mehrere CDM-Server parallel zur Verwaltung der Geräte eingesetzt werden können. Das CoAP Resource Directory besitzt einige Funktionen, die für die Registrierung der Geräte für dieses Konzept verwendet werden könnten. Daher sollte die Verwendung vom CoAP RD in Betracht gezogen werden, wenn die Spezifizierung abgeschlossen wurde. Ist dies der Fall, sollten auch die Methoden von Caturano et al. [CJR19], zum automatisierten Auffinden von CoAP-Endpunkten, in Betracht gezogen werden. Da die Registrierungen dann nicht mehr innerhalb des CDM-Servers verwaltet werden würden, führt dies auch zu der weiteren Überlegung, die übrige Funktionalität des Servers als Komponente innerhalb der MBP zu implementieren.

Das Konzept beschäftigt sich nicht mit dem physischen Standort der Geräte. Speziell für Sensoren oder Aktoren können Standortinformationen allerdings notwendig sein um den Kontext korrekt zu ermitteln. Daher sollte das Konzept um Methoden erweitert werden, um Standortinformationen der Geräte zu ermitteln.

Während der Implementierung kam es zu Kompatibilitätsproblemen zwischen der CoAP-Bibliotheken, die auf dem CDM-Server und auf den Geräten vom CDM-Agenten bzw. der Messaging Engine verwendet werden. Diese Probleme ließen sich auf eine nicht spezifikationskonforme Implementierung der clientseitigen Bibliothek zurückführen. Durch eine Änderung der serverseitigen Konfiguration, die dafür sorgt, dass problematische Teile der Nachrichten nach anderen Regeln erzeugt werden, konnten diese Probleme umgangen werden.

## Literaturverzeichnis

- [ABDP13] G. Aceto, A. Botta, W. De Donato, A. Pescapè. „Cloud monitoring: A survey“. In: *Computer Networks* 57.9 (2013), S. 2093–2115 (zitiert auf S. 42).
- [AIM10] L. Atzori, A. Iera, G. Morabito. „The Internet of Things: A survey“. In: *Computer Networks* 54.15 (Okt. 2010), S. 2787–2805. DOI: [10.1016/j.comnet.2010.05.010](https://doi.org/10.1016/j.comnet.2010.05.010) (zitiert auf S. 13).
- [BCS12] C. Bormann, A. P. Castellani, Z. Shelby. „CoAP: An Application Protocol for Billions of Tiny Internet Nodes“. In: *IEEE Internet Computing* 16.2 (März 2012), S. 62–67. DOI: [10.1109/mic.2012.29](https://doi.org/10.1109/mic.2012.29) (zitiert auf S. 20).
- [BFM05] T. Berners-Lee, R. T. Fielding, L. M. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. Jan. 2005. DOI: [10.17487/RFC3986](https://doi.org/10.17487/RFC3986). URL: <https://rfc-editor.org/rfc/rfc3986.txt> (zitiert auf S. 20).
- [BG14] A. Banks, R. Gupta. *MQTT Version 3.1.1*. Techn. Ber. OASIS Standard, 29. Okt. 2014. URL: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html> (zitiert auf S. 21).
- [BH13] C. Bormann, P. E. Hoffman. *Concise Binary Object Representation (CBOR)*. RFC 7049. Okt. 2013. DOI: [10.17487/RFC7049](https://doi.org/10.17487/RFC7049). URL: <https://rfc-editor.org/rfc/rfc7049.txt> (zitiert auf S. 47).
- [BJD16] J. Bugeja, A. Jacobsson, P. Davidsson. „On Privacy and Security Challenges in Smart Connected Homes“. In: *2016 European Intelligence and Security Informatics Conference (EISIC)*. Aug. 2016, S. 172–175. DOI: [10.1109/EISIC.2016.044](https://doi.org/10.1109/EISIC.2016.044) (zitiert auf S. 14).
- [BL98] F. Buschmann, C. Löckenhoff, Hrsg. *Pattern-orientierte Software-Architektur: ein Pattern-System*. Addison-Wesley-Longman, 1998. ISBN: 3827312825 (zitiert auf S. 18).
- [Bra17] T. Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259. Dez. 2017. DOI: [10.17487/RFC8259](https://doi.org/10.17487/RFC8259). URL: <https://rfc-editor.org/rfc/rfc8259.txt> (zitiert auf S. 47).
- [BSEB19] M. Breitbach, D. Schäfer, J. Edinger, C. Becker. „Context-aware data and task placement in edge computing environments“. In: *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2019 (zitiert auf S. 24).
- [CDF+02] B. Cain, D. S. E. Deering, B. Fenner, I. Kouvelas, A. Thyagarajan. *Internet Group Management Protocol, Version 3*. RFC 3376. Okt. 2002. DOI: [10.17487/RFC3376](https://doi.org/10.17487/RFC3376). URL: <https://rfc-editor.org/rfc/rfc3376.txt> (zitiert auf S. 18).

- [CJR19] F. Caturano, J. Jiménez, S. P. Romano. „Automated Discovery of CoAP-enabled IoT devices“. In: *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*. Juli 2019, S. 396–401. DOI: [10.1109/ICUFN.2019.8806084](https://doi.org/10.1109/ICUFN.2019.8806084) (zitiert auf S. 23, 62).
- [CYH+03] D. J. Cook, M. Youngblood, E. O. Heierman, K. Gopalratnam, S. Rao, A. Litvin, F. Khawaja. „MavHome: an agent-based smart home“. In: *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003)*. März 2003, S. 521–524. DOI: [10.1109/PERCOM.2003.1192783](https://doi.org/10.1109/PERCOM.2003.1192783) (zitiert auf S. 14).
- [DH19] D. Del Gaudio, P. Hirmer. „A lightweight messaging engine for decentralized data processing in the Internet of Things“. In: *SICS Software-Intensive Cyber-Physical Systems* (Aug. 2019). DOI: [10.1007/s00450-019-00410-z](https://doi.org/10.1007/s00450-019-00410-z) (zitiert auf S. 18, 19, 39, 40).
- [DRB+15] A. Donoho, B. Roe, M. Bodlaender, J. Gildred, A. Messer, Y. Kim, B. Fairman, J. Tourzan. *UPnP Device Architecture 2.0*. UPnP Forum, Feb. 2015. URL: <http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v2.0.pdf> (zitiert auf S. 24).
- [Dro97] R. Droms. *Dynamic Host Configuration Protocol*. RFC 2131. März 1997. DOI: [10.17487/RFC2131](https://doi.org/10.17487/RFC2131). URL: <https://rfc-editor.org/rfc/rfc2131.txt> (zitiert auf S. 31).
- [FHPM18] A. C. Franco da Silva, P. Hirmer, R. K. Peres, B. Mitschang. „An Approach for CEP Query Shipping to Support Distributed IoT Environments“. In: *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. März 2018, S. 247–252. DOI: [10.1109/PERCOMW.2018.8480241](https://doi.org/10.1109/PERCOMW.2018.8480241) (zitiert auf S. 21).
- [FT00] R. T. Fielding, R. N. Taylor. „Architectural Styles and the Design of Network-based Software Architectures“. AAI9980887. Diss. 2000. ISBN: 0-599-87118-0 (zitiert auf S. 20).
- [GBMP13] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami. „Internet of Things (IoT): A vision, architectural elements, and future directions“. In: *Future generation computer systems* 29.7 (2013), S. 1645–1660 (zitiert auf S. 37).
- [GRM17] A. Gupta, P. Rivana Christie, R. Manjula. „Scalability in Internet of Things: Features, Techniques and Research Challenges“. In: *International Journal of Computational Intelligence Research* 13.7 (2017), S. 1617–1627 (zitiert auf S. 13).
- [GTC16] S. Greene, H. Thapliyal, D. Carpenter. „IoT-Based Fall Detection for Smart Home Environments“. In: *2016 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS)*. Dez. 2016, S. 23–28. DOI: [10.1109/iNIS.2016.017](https://doi.org/10.1109/iNIS.2016.017) (zitiert auf S. 13).
- [Har03] R. Harper, Hrsg. *Inside the smart home*. Springer, 2003. ISBN: 1-85233-688-9 (zitiert auf S. 14).
- [HBS+16] P. Hirmer, U. Breitenbücher, A. C. F. da Silva, K. Képes, B. Mitschang, M. Wieland. „Automating the Provisioning and Configuration of Devices in the Internet of Things“. In: *Complex Systems Informatics and Modeling Quarterly* 9 (Dez. 2016), S. 28–43. DOI: [10.7250/csimq.2016-9.02](https://doi.org/10.7250/csimq.2016-9.02) (zitiert auf S. 21).



- [IKA+16] H. Ishii, K. Kimino, M. Aljehani, N. Ohe, M. Inoue. „An Early Detection System for Dementia Using the M2 M/IoT Platform“. In: *Procedia Computer Science* 96 (2016). Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 20th International Conference KES-2016, S. 1332–1340. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2016.08.178>. URL: <http://www.sciencedirect.com/science/article/pii/S1877050916319883> (zitiert auf S. 13).
- [JG15] Z. Jędrzejewski-Szmek, J. B. Gud. h. mundsson. „Daemon Management Under Systemd“. In: *login*: 40 (2015) (zitiert auf S. 48).
- [KB17] F. Knobloch, N. Braunschweig. „A Traffic-Aware Moving Light System Featuring Optimal Energy Efficiency“. In: *IEEE Sensors Journal* 17.23 (Dez. 2017), S. 7731–7740. DOI: [10.1109/JSEN.2017.2669398](https://doi.org/10.1109/JSEN.2017.2669398) (zitiert auf S. 13).
- [KLS14] M. Kovatsch, M. Lanter, Z. Shelby. „Californium: Scalable cloud services for the internet of things with coap“. In: *2014 International Conference on the Internet of Things (IOT)*. IEEE. 2014, S. 1–6 (zitiert auf S. 47).
- [LY06] C. M. Lonvick, T. Ylonen. *The Secure Shell (SSH) Connection Protocol*. RFC 4254. Jan. 2006. DOI: [10.17487/RFC4254](https://doi.org/10.17487/RFC4254). URL: <https://rfc-editor.org/rfc/rfc4254.txt> (zitiert auf S. 21).
- [Moc87] P. Mockapetris. *Domain names - concepts and facilities*. RFC 1034. Nov. 1987. DOI: [10.17487/RFC1034](https://doi.org/10.17487/RFC1034). URL: <https://rfc-editor.org/rfc/rfc1034.txt> (zitiert auf S. 32).
- [Mor17] R. Morabito. „Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation“. In: *IEEE Access* 5 (2017), S. 8835–8850. DOI: [10.1109/access.2017.2704444](https://doi.org/10.1109/access.2017.2704444) (zitiert auf S. 45).
- [Pat01] M. W. Patrick. *DHCP Relay Agent Information Option*. RFC 3046. Jan. 2001. DOI: [10.17487/RFC3046](https://doi.org/10.17487/RFC3046). URL: <https://rfc-editor.org/rfc/rfc3046.txt> (zitiert auf S. 32).
- [Pre02] R. Presuhn. *Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)*. RFC 3416. Dez. 2002. DOI: [10.17487/RFC3416](https://doi.org/10.17487/RFC3416). URL: <https://rfc-editor.org/rfc/rfc3416.txt> (zitiert auf S. 42).
- [RFC6690] Z. Shelby. *Constrained RESTful Environments (CoRE) Link Format*. RFC 6690. Aug. 2012. DOI: [10.17487/RFC6690](https://doi.org/10.17487/RFC6690). URL: <https://rfc-editor.org/rfc/rfc6690.txt> (zitiert auf S. 20, 49).
- [RFC7252] Z. Shelby, K. Hartke, C. Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252. Juni 2014. DOI: [10.17487/RFC7252](https://doi.org/10.17487/RFC7252). URL: <https://rfc-editor.org/rfc/rfc7252.txt> (zitiert auf S. 49, 58).
- [SCZ+16] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu. „Edge Computing: Vision and Challenges“. In: *IEEE Internet of Things Journal* 3.5 (Okt. 2016), S. 637–646. DOI: [10.1109/JIOT.2016.2579198](https://doi.org/10.1109/JIOT.2016.2579198) (zitiert auf S. 14).
- [Sel17] S. Selcuk. „Predictive maintenance, its implementation and latest trends“. In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 231.9 (2017), S. 1670–1679. DOI: [10.1177/0954405415601640](https://doi.org/10.1177/0954405415601640). eprint: <https://doi.org/10.1177/0954405415601640> (zitiert auf S. 13).

- [SKB+19] Z. Shelby, M. Koster, C. Bormann, P. V. der Stok, C. Amsüss. *CoRE Resource Directory*. Internet-Draft draft-ietf-core-resource-directory-23. Work in Progress. Internet Engineering Task Force, Juli 2019. 76 S. URL: <https://datatracker.ietf.org/doc/html/draft-ietf-core-resource-directory-23> (zitiert auf S. 23).
- [STXX13] X. Sheng, J. Tang, X. Xiao, G. Xue. „Sensing as a service: Challenges, solutions and future directions“. In: *IEEE Sensors journal* 13.10 (2013), S. 3733–3741 (zitiert auf S. 25).
- [TVM15] G. Tanganelli, C. Vallati, E. Mingozzi. „CoAPthon: Easy development of CoAP-based IoT applications with Python“. In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. Dez. 2015, S. 63–68. DOI: [10.1109/WF-IoT.2015.7389028](https://doi.org/10.1109/WF-IoT.2015.7389028) (zitiert auf S. 47).
- [VF13] O. Vermesan, P. Friess, Hrsg. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers Series in Communication. River publishers, 2013. ISBN: 978-87-92982-73-5 (zitiert auf S. 13).
- [YDZ+17] A. Yachir, B. Djamaa, K. Zeghouani, M. Bellal, M. Boudali. „Semantic Resource Discovery with CoAP in the Internet of Things“. In: Jan. 2017, S. 75–82. DOI: [10.5220/0006419400750082](https://doi.org/10.5220/0006419400750082) (zitiert auf S. 23).
- [YJX10] C. Yuqiang, G. Jianlan, H. Xuanzi. „The Research of Internet of Things’ Supporting Technologies Which Face the Logistics Industry“. In: *2010 International Conference on Computational Intelligence and Security*. Dez. 2010, S. 659–663. DOI: [10.1109/CIS.2010.148](https://doi.org/10.1109/CIS.2010.148) (zitiert auf S. 13).

Alle URLs wurden zuletzt am 25. 10. 2019 geprüft.

### **Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

Ort, Datum, Unterschrift